```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <string.h>

#include <shastra/datacomm/shastraDataH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

int
shaCharOut(fd, pShaChar)
    int             fd;
    shaChar      *pShaChar;
{
    if(!xdr_shaChar(mplexXDRSEnc(fd), pShaChar)){
        return -1;
    }
    return 1;
}

int
shaCharIn(fd, pShaChar)
    int             fd;
    shaChar      *pShaChar;
{
    if(!xdr_shaChar(mplexXDRSDec(fd), pShaChar)){
        return -1;
    }
    return 1;
}
```

```
int
shaCharsOut(fd, pShaChars)
    int             fd;
    shaChars     *pShaChars;
{
    if(!xdr_shaChars(mplexXDRSEnc(fd), pShaChars)){
        return  -1;
    }
    return 1;
}

int
shaCharsIn(fd, pShaChars)
    int             fd;
    shaChars     *pShaChars;
{
    shaCharsXDRFree(pShaChars);
    if(!xdr_shaChars(mplexXDRSDec(fd), pShaChars)){
        return -1;
    }
    return 1;
}


void
freeShaChars(pShaChars)
    shaChars     *pShaChars;
{
    int             i;

    if (pShaChars == NULL) {
        return;
    }
    free(pShaChars->shaChars_val);
    memset(pShaChars, 0,sizeof(shaChars));
}

shaChars     *
copyShaChars(pShaChars, destpShaChars)
    shaChars     *pShaChars;
    shaChars     *destpShaChars;
{
    int             i;
    shaChars     *newpShaChars;

    if (pShaChars == NULL) {
        return NULL;
    }
    if (destpShaChars == NULL) {
        newpShaChars = (shaChars *) malloc(sizeof(shaChars));
    } else {
        newpShaChars = destpShaChars;
    }
```

```
    memcpy(newpShaChars, pShaChars, sizeof(shaChars));
    newpShaChars->shaChars_val = (shaChar *)
        malloc(newpShaChars->shaChars_len * sizeof(shaChar));
    memcpy(newpShaChars->shaChars_val, pShaChars->shaChars_val,
        pShaChars->shaChars_len * sizeof(shaChar));
    return newpShaChars;
}

void
inputShaChars(fp, pShaChars)
    FILE           *fp;
    shaChars     *pShaChars;
{
    int             i;

    fscanf(fp, "%d", &pShaChars->shaChars_len);
    pShaChars->shaChars_val = (shaChar *)
        malloc(pShaChars->shaChars_len * sizeof(shaChar));
    for (i = 0; i < pShaChars->shaChars_len; i++) {
        pShaChars->shaChars_val[i] = fgetc(fp);
    }
}

void
outputShaChars(fp, pShaChars)
    FILE           *fp;
    shaChars     *pShaChars;
{
    int             i;

    fprintf(fp, "%d\n", pShaChars->shaChars_len);
    for (i = 0; i < pShaChars->shaChars_len; i++) {
        fputc(pShaChars->shaChars_val[i], fp);
    }

}

void
shaCharsXDRFree(pShaChars)
    shaChars     *pShaChars;
{
    xdr_free(xdr_shaChars, (char *) pShaChars);
    memset(pShaChars, 0,sizeof(shaChars));
}


int
shaUCharOut(fd, pShaUChar)
    int             fd;
    shaUChar      *pShaUChar;
{
    if(!xdr_shaUChar(mplexXDRSEnc(fd), pShaUChar)){
        return -1;
```

```
    }
    return 1;
}

int
shaUCharIn(fd, pShaUChar)
    int             fd;
    shaUChar      *pShaUChar;
{
    if(!xdr_shaUChar(mplexXDRSDec(fd), pShaUChar)){
        return -1;
    }
    return 1;
}

int
shaUCharsOut(fd, pShaUChars)
    int             fd;
    shaUChars     *pShaUChars;
{
    if(!xdr_shaUChars(mplexXDRSEnc(fd), pShaUChars)){
        return  -1;
    }
    return 1;
}

int
shaUCharsIn(fd, pShaUChars)
    int             fd;
    shaUChars     *pShaUChars;
{
    shaUCharsXDRFree(pShaUChars);
    if(!xdr_shaUChars(mplexXDRSDec(fd), pShaUChars)){
        return -1;
    }
    return 1;
}


void
freeShaUChars(pShaUChars)
    shaUChars     *pShaUChars;
{
    int             i;

    if (pShaUChars == NULL) {
        return;
    }
    free(pShaUChars->shaUChars_val);
    memset(pShaUChars, 0,sizeof(shaUChars));
}

shaUChars     *
```

```
copyShaUChars(pShaUChars, destpShaUChars)
    shaUChars     *pShaUChars;
    shaUChars     *destpShaUChars;
{
    int             i;
    shaUChars     *newpShaUChars;

    if (pShaUChars == NULL) {
        return NULL;
    }
    if (destpShaUChars == NULL) {
        newpShaUChars = (shaUChars *) malloc(sizeof(shaUChars));
    } else {
        newpShaUChars = destpShaUChars;
    }
    memcpy(newpShaUChars, pShaUChars, sizeof(shaUChars));
    newpShaUChars->shaUChars_val = (shaUChar *)
        malloc(newpShaUChars->shaUChars_len * sizeof(shaUChar));
    memcpy(newpShaUChars->shaUChars_val, pShaUChars->shaUChars_val,
        pShaUChars->shaUChars_len * sizeof(shaUChar));
    return newpShaUChars;
}

void
inputShaUChars(fp, pShaUChars)
    FILE           *fp;
    shaUChars     *pShaUChars;
{
    int             i;

    fscanf(fp, "%d", &pShaUChars->shaUChars_len);
    pShaUChars->shaUChars_val = (shaUChar *)
        malloc(pShaUChars->shaUChars_len * sizeof(shaUChar));
    for (i = 0; i < pShaUChars->shaUChars_len; i++) {
        pShaUChars->shaUChars_val[i] = fgetc(fp);
    }
}

void
outputShaUChars(fp, pShaUChars)
    FILE           *fp;
    shaUChars     *pShaUChars;
{
    int             i;

    fprintf(fp, "%d\n", pShaUChars->shaUChars_len);
    for (i = 0; i < pShaUChars->shaUChars_len; i++) {
        fputc(pShaUChars->shaUChars_val[i], fp);
    }

}

void
```

```
shaUCharsXDRFree(pShaUChars)
    shaUChars     *pShaUChars;
{
    xdr_free(xdr_shaUChars, (char *) pShaUChars);
    memset(pShaUChars, 0,sizeof(shaUChars));
}

int
shaShortOut(fd, pShaShort)
    int             fd;
    shaShort      *pShaShort;
{
    if(!xdr_shaShort(mplexXDRSEnc(fd), pShaShort)){
        return -1;
    }
    return 1;
}

int
shaShortIn(fd, pShaShort)
    int             fd;
    shaShort      *pShaShort;
{
    if(!xdr_shaShort(mplexXDRSDec(fd), pShaShort)){
        return -1;
    }
    return 1;
}

int
shaShortsOut(fd, pShaShorts)
    int             fd;
    shaShorts     *pShaShorts;
{
    if(!xdr_shaShorts(mplexXDRSEnc(fd), pShaShorts)){
        return  -1;
    }
    return 1;
}

int
shaShortsIn(fd, pShaShorts)
    int             fd;
    shaShorts     *pShaShorts;
{
    shaShortsXDRFree(pShaShorts);
    if(!xdr_shaShorts(mplexXDRSDec(fd), pShaShorts)){
        return -1;
    }
    return 1;
}
```

```
void
freeShaShorts(pShaShorts)
    shaShorts     *pShaShorts;
{
    int             i;

    if (pShaShorts == NULL) {
        return;
    }
    free(pShaShorts->shaShorts_val);
    memset(pShaShorts,0,sizeof(shaShorts));
}

shaShorts     *
copyShaShorts(pShaShorts, destpShaShorts)
    shaShorts     *pShaShorts;
    shaShorts     *destpShaShorts;
{
    int             i;
    shaShorts     *newpShaShorts;

    if (pShaShorts == NULL) {
        return NULL;
    }
    if (destpShaShorts == NULL) {
        newpShaShorts = (shaShorts *) malloc(sizeof(shaShorts));
    } else {
        newpShaShorts = destpShaShorts;
    }
    memcpy(newpShaShorts, pShaShorts, sizeof(shaShorts));
    newpShaShorts->shaShorts_val = (shaShort *)
        malloc(newpShaShorts->shaShorts_len * sizeof(shaShort));
    memcpy(newpShaShorts->shaShorts_val, pShaShorts->shaShorts_val,
        pShaShorts->shaShorts_len * sizeof(shaShort));
    return newpShaShorts;
}

void
inputShaShorts(fp, pShaShorts)
    FILE           *fp;
    shaShorts     *pShaShorts;
{
    int             i;

    fscanf(fp, "%d", &pShaShorts->shaShorts_len);
    pShaShorts->shaShorts_val = (shaShort *)
        malloc(pShaShorts->shaShorts_len * sizeof(shaShort));
    for (i = 0; i < pShaShorts->shaShorts_len; i++) {
        fscanf(fp,"%h", &pShaShorts->shaShorts_val[i]);
    }
}

void
```

```
outputShaShorts(fp, pShaShorts)
    FILE           *fp;
    shaShorts     *pShaShorts;
{
    int             i;

    fprintf(fp, "%d\n", pShaShorts->shaShorts_len);
    for (i = 0; i < pShaShorts->shaShorts_len; i++) {
        fprintf(fp,"%h", pShaShorts->shaShorts_val[i]);
    }

}

void
shaShortsXDRFree(pShaShorts)
    shaShorts     *pShaShorts;
{
    xdr_free(xdr_shaShorts, (char *) pShaShorts);
    memset(pShaShorts,0,sizeof(shaShorts));
}

int
shaUShortOut(fd, pShaUShort)
    int             fd;
    shaUShort      *pShaUShort;
{
    if(!xdr_shaUShort(mplexXDRSEnc(fd), pShaUShort)){
        return -1;
    }
    return 1;
}

int
shaUShortIn(fd, pShaUShort)
    int             fd;
    shaUShort      *pShaUShort;
{
    if(!xdr_shaUShort(mplexXDRSDec(fd), pShaUShort)){
        return -1;
    }
    return 1;
}

int
shaUShortsOut(fd, pShaUShorts)
    int             fd;
    shaUShorts     *pShaUShorts;
{
    if(!xdr_shaUShorts(mplexXDRSEnc(fd), pShaUShorts)){
        return  -1;
    }
    return 1;
}
```

```
int
shaUShortsIn(fd, pShaUShorts)
    int             fd;
    shaUShorts     *pShaUShorts;
{
    shaUShortsXDRFree(pShaUShorts);
    if(!xdr_shaUShorts(mplexXDRSDec(fd), pShaUShorts)){
        return -1;
    }
    return 1;
}


void
freeShaUShorts(pShaUShorts)
    shaUShorts     *pShaUShorts;
{
    int             i;

    if (pShaUShorts == NULL) {
        return;
    }
    free(pShaUShorts->shaUShorts_val);
    memset(pShaUShorts,0,sizeof(shaUShorts));
}

shaUShorts     *
copyShaUShorts(pShaUShorts, destpShaUShorts)
    shaUShorts     *pShaUShorts;
    shaUShorts     *destpShaUShorts;
{
    int             i;
    shaUShorts     *newpShaUShorts;

    if (pShaUShorts == NULL) {
        return NULL;
    }
    if (destpShaUShorts == NULL) {
        newpShaUShorts = (shaUShorts *) malloc(sizeof(shaUShorts));
    } else {
        newpShaUShorts = destpShaUShorts;
    }
    memcpy(newpShaUShorts, pShaUShorts, sizeof(shaUShorts));
    newpShaUShorts->shaUShorts_val = (shaUShort *)
        malloc(newpShaUShorts->shaUShorts_len * sizeof(shaUShort));
    memcpy(newpShaUShorts->shaUShorts_val, pShaUShorts->shaUShorts_val,
        pShaUShorts->shaUShorts_len * sizeof(shaUShort));
    return newpShaUShorts;
}

void
inputShaUShorts(fp, pShaUShorts)
```

```
    FILE           *fp;
    shaUShorts     *pShaUShorts;
{
    int             i;

    fscanf(fp, "%d", &pShaUShorts->shaUShorts_len);
    pShaUShorts->shaUShorts_val = (shaUShort *)
        malloc(pShaUShorts->shaUShorts_len * sizeof(shaUShort));
    for (i = 0; i < pShaUShorts->shaUShorts_len; i++) {
        fscanf(fp,"%h", &pShaUShorts->shaUShorts_val[i]);
    }
}

void
outputShaUShorts(fp, pShaUShorts)
    FILE           *fp;
    shaUShorts     *pShaUShorts;
{
    int             i;

    fprintf(fp, "%d\n", pShaUShorts->shaUShorts_len);
    for (i = 0; i < pShaUShorts->shaUShorts_len; i++) {
        fprintf(fp,"%h", pShaUShorts->shaUShorts_val[i]);
    }

}

void
shaUShortsXDRFree(pShaUShorts)
    shaUShorts     *pShaUShorts;
{
    xdr_free(xdr_shaUShorts, (char *) pShaUShorts);
    memset(pShaUShorts,0,sizeof(shaUShorts));
}

int
shaIntOut(fd, pShaInt)
    int             fd;
    shaInt      *pShaInt;
{
    if(!xdr_shaInt(mplexXDRSEnc(fd), pShaInt)){
        return -1;
    }
    return 1;
}

int
shaIntIn(fd, pShaInt)
    int             fd;
    shaInt      *pShaInt;
{
    if(!xdr_shaInt(mplexXDRSDec(fd), pShaInt)){
        return -1;
```

```
    }
    return 1;
}

int
shaIntsOut(fd, pShaInts)
    int             fd;
    shaInts     *pShaInts;
{
    if(!xdr_shaInts(mplexXDRSEnc(fd), pShaInts)){
        return  -1;
    }
    return 1;
}

int
shaIntsIn(fd, pShaInts)
    int             fd;
    shaInts     *pShaInts;
{
    shaIntsXDRFree(pShaInts);
    if(!xdr_shaInts(mplexXDRSDec(fd), pShaInts)){
        return -1;
    }
    return 1;
}


void
freeShaInts(pShaInts)
    shaInts     *pShaInts;
{
    int             i;

    if (pShaInts == NULL) {
        return;
    }
    free(pShaInts->shaInts_val);
    memset(pShaInts,0,sizeof(shaInts));
}

shaInts     *
copyShaInts(pShaInts, destpShaInts)
    shaInts     *pShaInts;
    shaInts     *destpShaInts;
{
    int             i;
    shaInts     *newpShaInts;

    if (pShaInts == NULL) {
        return NULL;
    }
    if (destpShaInts == NULL) {
```

```
        newpShaInts = (shaInts *) malloc(sizeof(shaInts));
    } else {
        newpShaInts = destpShaInts;
    }
    memcpy(newpShaInts, pShaInts, sizeof(shaInts));
    newpShaInts->shaInts_val = (shaInt *)
        malloc(newpShaInts->shaInts_len * sizeof(shaInt));
    memcpy(newpShaInts->shaInts_val, pShaInts->shaInts_val,
        pShaInts->shaInts_len * sizeof(shaInt));
    return newpShaInts;
}

void
inputShaInts(fp, pShaInts)
    FILE           *fp;
    shaInts     *pShaInts;
{
    int             i;

    fscanf(fp, "%d", &pShaInts->shaInts_len);
    pShaInts->shaInts_val = (shaInt *)
        malloc(pShaInts->shaInts_len * sizeof(shaInt));
    for (i = 0; i < pShaInts->shaInts_len; i++) {
        fscanf(fp,"%h", &pShaInts->shaInts_val[i]);
    }
}

void
outputShaInts(fp, pShaInts)
    FILE           *fp;
    shaInts     *pShaInts;
{
    int             i;

    fprintf(fp, "%d\n", pShaInts->shaInts_len);
    for (i = 0; i < pShaInts->shaInts_len; i++) {
        fprintf(fp,"%h", pShaInts->shaInts_val[i]);
    }

}

void
shaIntsXDRFree(pShaInts)
    shaInts     *pShaInts;
{
    xdr_free(xdr_shaInts, (char *) pShaInts);
    memset(pShaInts,0,sizeof(shaInts));
}

int
shaUIntOut(fd, pShaUInt)
    int             fd;
    shaUInt      *pShaUInt;
```

```
{
    if(!xdr_shaUInt(mplexXDRSEnc(fd), pShaUInt)){
        return -1;
    }
    return 1;
}

int
shaUIntIn(fd, pShaUInt)
    int             fd;
    shaUInt      *pShaUInt;
{
    if(!xdr_shaUInt(mplexXDRSDec(fd), pShaUInt)){
        return -1;
    }
    return 1;
}

int
shaUIntsOut(fd, pShaUInts)
    int             fd;
    shaUInts     *pShaUInts;
{
    if(!xdr_shaUInts(mplexXDRSEnc(fd), pShaUInts)){
        return  -1;
    }
    return 1;
}

int
shaUIntsIn(fd, pShaUInts)
    int             fd;
    shaUInts     *pShaUInts;
{
    shaUIntsXDRFree(pShaUInts);
    if(!xdr_shaUInts(mplexXDRSDec(fd), pShaUInts)){
        return -1;
    }
    return 1;
}


void
freeShaUInts(pShaUInts)
    shaUInts     *pShaUInts;
{
    int             i;

    if (pShaUInts == NULL) {
        return;
    }
    free(pShaUInts->shaUInts_val);
    memset(pShaUInts,0,sizeof(shaUInts));
```

```
}

shaUInts     *
copyShaUInts(pShaUInts, destpShaUInts)
    shaUInts     *pShaUInts;
    shaUInts     *destpShaUInts;
{
    int             i;
    shaUInts     *newpShaUInts;

    if (pShaUInts == NULL) {
        return NULL;
    }
    if (destpShaUInts == NULL) {
        newpShaUInts = (shaUInts *) malloc(sizeof(shaUInts));
    } else {
        newpShaUInts = destpShaUInts;
    }
    memcpy(newpShaUInts, pShaUInts, sizeof(shaUInts));
    newpShaUInts->shaUInts_val = (shaUInt *)
        malloc(newpShaUInts->shaUInts_len * sizeof(shaUInt));
    memcpy(newpShaUInts->shaUInts_val, pShaUInts->shaUInts_val,
        pShaUInts->shaUInts_len * sizeof(shaUInt));
    return newpShaUInts;
}

void
inputShaUInts(fp, pShaUInts)
    FILE           *fp;
    shaUInts     *pShaUInts;
{
    int             i;

    fscanf(fp, "%d", &pShaUInts->shaUInts_len);
    pShaUInts->shaUInts_val = (shaUInt *)
        malloc(pShaUInts->shaUInts_len * sizeof(shaUInt));
    for (i = 0; i < pShaUInts->shaUInts_len; i++) {
        fscanf(fp,"%h", &pShaUInts->shaUInts_val[i]);
    }
}

void
outputShaUInts(fp, pShaUInts)
    FILE           *fp;
    shaUInts     *pShaUInts;
{
    int             i;

    fprintf(fp, "%d\n", pShaUInts->shaUInts_len);
    for (i = 0; i < pShaUInts->shaUInts_len; i++) {
        fprintf(fp,"%h", pShaUInts->shaUInts_val[i]);
    }
```

```
}

void
shaUIntsXDRFree(pShaUInts)
    shaUInts     *pShaUInts;
{
    xdr_free(xdr_shaUInts, (char *) pShaUInts);
    memset(pShaUInts,0,sizeof(shaUInts));
}

int
shaLongOut(fd, pShaLong)
    int             fd;
    shaLong      *pShaLong;
{
    if(!xdr_shaLong(mplexXDRSEnc(fd), pShaLong)){
        return -1;
    }
    return 1;
}

int
shaLongIn(fd, pShaLong)
    int             fd;
    shaLong      *pShaLong;
{
    if(!xdr_shaLong(mplexXDRSDec(fd), pShaLong)){
        return -1;
    }
    return 1;
}

int
shaLongsOut(fd, pShaLongs)
    int             fd;
    shaLongs     *pShaLongs;
{
    if(!xdr_shaLongs(mplexXDRSEnc(fd), pShaLongs)){
        return  -1;
    }
    return 1;
}

int
shaLongsIn(fd, pShaLongs)
    int             fd;
    shaLongs     *pShaLongs;
{
    shaLongsXDRFree(pShaLongs);
    if(!xdr_shaLongs(mplexXDRSDec(fd), pShaLongs)){
        return -1;
    }
    return 1;
```

```
}


void
freeShaLongs(pShaLongs)
    shaLongs     *pShaLongs;
{
    int             i;

    if (pShaLongs == NULL) {
        return;
    }
    free(pShaLongs->shaLongs_val);
    memset(pShaLongs,0,sizeof(shaLongs));
}

shaLongs     *
copyShaLongs(pShaLongs, destpShaLongs)
    shaLongs     *pShaLongs;
    shaLongs     *destpShaLongs;
{
    int             i;
    shaLongs     *newpShaLongs;

    if (pShaLongs == NULL) {
        return NULL;
    }
    if (destpShaLongs == NULL) {
        newpShaLongs = (shaLongs *) malloc(sizeof(shaLongs));
    } else {
        newpShaLongs = destpShaLongs;
    }
    memcpy(newpShaLongs, pShaLongs, sizeof(shaLongs));
    newpShaLongs->shaLongs_val = (shaLong *)
        malloc(newpShaLongs->shaLongs_len * sizeof(shaLong));
    memcpy(newpShaLongs->shaLongs_val, pShaLongs->shaLongs_val,
        pShaLongs->shaLongs_len * sizeof(shaLong));
    return newpShaLongs;
}

void
inputShaLongs(fp, pShaLongs)
    FILE           *fp;
    shaLongs     *pShaLongs;
{
    int             i;

    fscanf(fp, "%d", &pShaLongs->shaLongs_len);
    pShaLongs->shaLongs_val = (shaLong *)
        malloc(pShaLongs->shaLongs_len * sizeof(shaLong));
    for (i = 0; i < pShaLongs->shaLongs_len; i++) {
        fscanf(fp,"%h", &pShaLongs->shaLongs_val[i]);
    }
```

```
}

void
outputShaLongs(fp, pShaLongs)
    FILE           *fp;
    shaLongs     *pShaLongs;
{
    int             i;

    fprintf(fp, "%d\n", pShaLongs->shaLongs_len);
    for (i = 0; i < pShaLongs->shaLongs_len; i++) {
        fprintf(fp,"%h", pShaLongs->shaLongs_val[i]);
    }

}

void
shaLongsXDRFree(pShaLongs)
    shaLongs     *pShaLongs;
{
    xdr_free(xdr_shaLongs, (char *) pShaLongs);
    memset(pShaLongs,0,sizeof(shaLongs));
}

int
shaULongOut(fd, pShaULong)
    int             fd;
    shaULong      *pShaULong;
{
    if(!xdr_shaULong(mplexXDRSEnc(fd), pShaULong)){
        return -1;
    }
    return 1;
}

int
shaULongIn(fd, pShaULong)
    int             fd;
    shaULong      *pShaULong;
{
    if(!xdr_shaULong(mplexXDRSDec(fd), pShaULong)){
        return -1;
    }
    return 1;
}

int
shaULongsOut(fd, pShaULongs)
    int             fd;
    shaULongs     *pShaULongs;
{
    if(!xdr_shaULongs(mplexXDRSEnc(fd), pShaULongs)){
        return  -1;
```

```
    }
    return 1;
}

int
shaULongsIn(fd, pShaULongs)
    int             fd;
    shaULongs     *pShaULongs;
{
    shaULongsXDRFree(pShaULongs);
    if(!xdr_shaULongs(mplexXDRSDec(fd), pShaULongs)){
        return -1;
    }
    return 1;
}


void
freeShaULongs(pShaULongs)
    shaULongs     *pShaULongs;
{
    int             i;

    if (pShaULongs == NULL) {
        return;
    }
    free(pShaULongs->shaULongs_val);
    memset(pShaULongs,0,sizeof(shaULongs));
}

shaULongs     *
copyShaULongs(pShaULongs, destpShaULongs)
    shaULongs     *pShaULongs;
    shaULongs     *destpShaULongs;
{
    int             i;
    shaULongs     *newpShaULongs;

    if (pShaULongs == NULL) {
        return NULL;
    }
    if (destpShaULongs == NULL) {
        newpShaULongs = (shaULongs *) malloc(sizeof(shaULongs));
    } else {
        newpShaULongs = destpShaULongs;
    }
    memcpy(newpShaULongs, pShaULongs, sizeof(shaULongs));
    newpShaULongs->shaULongs_val = (shaULong *)
        malloc(newpShaULongs->shaULongs_len * sizeof(shaULong));
    memcpy(newpShaULongs->shaULongs_val, pShaULongs->shaULongs_val,
        pShaULongs->shaULongs_len * sizeof(shaULong));
    return newpShaULongs;
}
```

```
void
inputShaULongs(fp, pShaULongs)
    FILE           *fp;
    shaULongs     *pShaULongs;
{
    int             i;

    fscanf(fp, "%d", &pShaULongs->shaULongs_len);
    pShaULongs->shaULongs_val = (shaULong *)
        malloc(pShaULongs->shaULongs_len * sizeof(shaULong));
    for (i = 0; i < pShaULongs->shaULongs_len; i++) {
        fscanf(fp,"%h", &pShaULongs->shaULongs_val[i]);
    }
}

void
outputShaULongs(fp, pShaULongs)
    FILE           *fp;
    shaULongs     *pShaULongs;
{
    int             i;

    fprintf(fp, "%d\n", pShaULongs->shaULongs_len);
    for (i = 0; i < pShaULongs->shaULongs_len; i++) {
        fprintf(fp,"%h", pShaULongs->shaULongs_val[i]);
    }

}

void
shaULongsXDRFree(pShaULongs)
    shaULongs     *pShaULongs;
{
    xdr_free(xdr_shaULongs, (char *) pShaULongs);
    memset(pShaULongs,0,sizeof(shaULongs));
}

int
shaFloatOut(fd, pShaFloat)
    int             fd;
    shaFloat      *pShaFloat;
{
    if(!xdr_shaFloat(mplexXDRSEnc(fd), pShaFloat)){
        return -1;
    }
    return 1;
}

int
shaFloatIn(fd, pShaFloat)
    int             fd;
    shaFloat      *pShaFloat;
```

```
{
    if(!xdr_shaFloat(mplexXDRSDec(fd), pShaFloat)){
        return -1;
    }
    return 1;
}

int
shaFloatsOut(fd, pShaFloats)
    int             fd;
    shaFloats     *pShaFloats;
{
    if(!xdr_shaFloats(mplexXDRSEnc(fd), pShaFloats)){
        return  -1;
    }
    return 1;
}

int
shaFloatsIn(fd, pShaFloats)
    int             fd;
    shaFloats     *pShaFloats;
{
    shaFloatsXDRFree(pShaFloats);
    if(!xdr_shaFloats(mplexXDRSDec(fd), pShaFloats)){
        return -1;
    }
    return 1;
}


void
freeShaFloats(pShaFloats)
    shaFloats     *pShaFloats;
{
    int             i;

    if (pShaFloats == NULL) {
        return;
    }
    free(pShaFloats->shaFloats_val);
    memset(pShaFloats,0,sizeof(shaFloats));
}

shaFloats     *
copyShaFloats(pShaFloats, destpShaFloats)
    shaFloats     *pShaFloats;
    shaFloats     *destpShaFloats;
{
    int             i;
    shaFloats     *newpShaFloats;

    if (pShaFloats == NULL) {
```

```
        return NULL;
    }
    if (destpShaFloats == NULL) {
        newpShaFloats = (shaFloats *) malloc(sizeof(shaFloats));
    } else {
        newpShaFloats = destpShaFloats;
    }
    memcpy(newpShaFloats, pShaFloats, sizeof(shaFloats));
    newpShaFloats->shaFloats_val = (shaFloat *)
        malloc(newpShaFloats->shaFloats_len * sizeof(shaFloat));
    memcpy(newpShaFloats->shaFloats_val, pShaFloats->shaFloats_val,
        pShaFloats->shaFloats_len * sizeof(shaFloat));
    return newpShaFloats;
}

void
inputShaFloats(fp, pShaFloats)
    FILE           *fp;
    shaFloats     *pShaFloats;
{
    int             i;

    fscanf(fp, "%d", &pShaFloats->shaFloats_len);
    pShaFloats->shaFloats_val = (shaFloat *)
        malloc(pShaFloats->shaFloats_len * sizeof(shaFloat));
    for (i = 0; i < pShaFloats->shaFloats_len; i++) {
        fscanf(fp,"%h", &pShaFloats->shaFloats_val[i]);
    }
}

void
outputShaFloats(fp, pShaFloats)
    FILE           *fp;
    shaFloats     *pShaFloats;
{
    int             i;

    fprintf(fp, "%d\n", pShaFloats->shaFloats_len);
    for (i = 0; i < pShaFloats->shaFloats_len; i++) {
        fprintf(fp,"%h", pShaFloats->shaFloats_val[i]);
    }

}

void
shaFloatsXDRFree(pShaFloats)
    shaFloats     *pShaFloats;
{
    xdr_free(xdr_shaFloats, (char *) pShaFloats);
    memset(pShaFloats,0,sizeof(shaFloats));
}

int
```

```
shaDoubleOut(fd, pShaDouble)
    int              fd;
    shaDouble      *pShaDouble;
{
    if(!xdr_shaDouble(mplexXDRSEnc(fd), pShaDouble)){
        return -1;
    }
    return 1;
}

int
shaDoubleIn(fd, pShaDouble)
    int              fd;
    shaDouble      *pShaDouble;
{
    if(!xdr_shaDouble(mplexXDRSDec(fd), pShaDouble)){
        return -1;
    }
    return 1;
}

int
shaDoublesOut(fd, pShaDoubles)
    int              fd;
    shaDoubles     *pShaDoubles;
{
    if(!xdr_shaDoubles(mplexXDRSEnc(fd), pShaDoubles)){
        return  -1;
    }
    return 1;
}

int
shaDoublesIn(fd, pShaDoubles)
    int              fd;
    shaDoubles     *pShaDoubles;
{
    shaDoublesXDRFree(pShaDoubles);
    if(!xdr_shaDoubles(mplexXDRSDec(fd), pShaDoubles)){
        return -1;
    }
    return 1;
}


void
freeShaDoubles(pShaDoubles)
    shaDoubles     *pShaDoubles;
{
    int              i;

    if (pShaDoubles == NULL) {
        return;
```

```
    }
    free(pShaDoubles->shaDoubles_val);
    memset(pShaDoubles,0,sizeof(shaDoubles));
}

shaDoubles     *
copyShaDoubles(pShaDoubles, destpShaDoubles)
    shaDoubles     *pShaDoubles;
    shaDoubles     *destpShaDoubles;
{
    int             i;
    shaDoubles     *newpShaDoubles;

    if (pShaDoubles == NULL) {
        return NULL;
    }
    if (destpShaDoubles == NULL) {
        newpShaDoubles = (shaDoubles *) malloc(sizeof(shaDoubles));
    } else {
        newpShaDoubles = destpShaDoubles;
    }
    memcpy(newpShaDoubles, pShaDoubles, sizeof(shaDoubles));
    newpShaDoubles->shaDoubles_val = (shaDouble *)
        malloc(newpShaDoubles->shaDoubles_len * sizeof(shaDouble));
    memcpy(newpShaDoubles->shaDoubles_val, pShaDoubles->shaDoubles_val,
        pShaDoubles->shaDoubles_len * sizeof(shaDouble));
    return newpShaDoubles;
}

void
inputShaDoubles(fp, pShaDoubles)
    FILE           *fp;
    shaDoubles     *pShaDoubles;
{
    int             i;

    fscanf(fp, "%d", &pShaDoubles->shaDoubles_len);
    pShaDoubles->shaDoubles_val = (shaDouble *)
        malloc(pShaDoubles->shaDoubles_len * sizeof(shaDouble));
    for (i = 0; i < pShaDoubles->shaDoubles_len; i++) {
        fscanf(fp,"%h", &pShaDoubles->shaDoubles_val[i]);
    }
}

void
outputShaDoubles(fp, pShaDoubles)
    FILE           *fp;
    shaDoubles     *pShaDoubles;
{
    int             i;

    fprintf(fp, "%d\n", pShaDoubles->shaDoubles_len);
    for (i = 0; i < pShaDoubles->shaDoubles_len; i++) {
```

```
        fprintf(fp,"%h", pShaDoubles->shaDoubles_val[i]);
    }

}

void
shaDoublesXDRFree(pShaDoubles)
    shaDoubles     *pShaDoubles;
{
    xdr_free(xdr_shaDoubles, (char *) pShaDoubles);
    memset(pShaDoubles,0,sizeof(shaDoubles));
}

int
shaStringOut(fd, pShaString)
    int             fd;
    shaString      *pShaString;
{
    if(!xdr_shaString(mplexXDRSEnc(fd), pShaString)){
        return -1;
    }
    return 1;
}

int
shaStringIn(fd, pShaString)
    int             fd;
    shaString      *pShaString;
{
    if(!xdr_shaString(mplexXDRSDec(fd), pShaString)){
        return -1;
    }
    return 1;
}

int
shaStringsOut(fd, pShaStrings)
    int             fd;
    shaStrings     *pShaStrings;
{
    if(!xdr_shaStrings(mplexXDRSEnc(fd), pShaStrings)){
        return  -1;
    }
    return 1;
}

int
shaStringsIn(fd, pShaStrings)
    int             fd;
    shaStrings     *pShaStrings;
{
    shaStringsXDRFree(pShaStrings);
    if(!xdr_shaStrings(mplexXDRSDec(fd), pShaStrings)){
```

```
        return -1;
    }
    return 1;
}

void
freeShaStrings(pShaStrings)
    shaStrings     *pShaStrings;
{
    int             i;

    if (pShaStrings == NULL) {
        return;
    }
    for(i=0;i<pShaStrings->shaStrings_len;i++){
        free(pShaStrings->shaStrings_val[i]);
    }
    free(pShaStrings->shaStrings_val);
    memset(pShaStrings,0,sizeof(shaStrings));
}

shaStrings     *
copyShaStrings(pShaStrings, destpShaStrings)
    shaStrings     *pShaStrings;
    shaStrings     *destpShaStrings;
{
    int             i;
    shaStrings     *newpShaStrings;

    if (pShaStrings == NULL) {
        return NULL;
    }
    if (destpShaStrings == NULL) {
        newpShaStrings = (shaStrings *) malloc(sizeof(shaStrings));
    } else {
        newpShaStrings = destpShaStrings;
    }
    memcpy(newpShaStrings, pShaStrings, sizeof(shaStrings));
    newpShaStrings->shaStrings_val = (shaString *)
        malloc(newpShaStrings->shaStrings_len * sizeof(shaString));
    for(i=0; i<pShaStrings->shaStrings_len;i++){
        newpShaStrings->shaStrings_val[i] =
            strdup(pShaStrings->shaStrings_val[i]);
    }
    return newpShaStrings;
}

void
inputShaStrings(fp, pShaStrings)
    FILE           *fp;
    shaStrings     *pShaStrings;
{
    int             i;
```

```
    fscanf(fp, "%d", &pShaStrings->shaStrings_len);
    pShaStrings->shaStrings_val = (shaString *)
        malloc(pShaStrings->shaStrings_len * sizeof(shaString));
    for (i = 0; i < pShaStrings->shaStrings_len; i++) {
        pShaStrings->shaStrings_val[i] = malloc(1024);
        fgets(pShaStrings->shaStrings_val[i], 1024,fp);
        pShaStrings->shaStrings_val[i] =
            realloc(pShaStrings->shaStrings_val[i],
                strlen(pShaStrings->shaStrings_val[i]));
    }
}

void
outputShaStrings(fp, pShaStrings)
    FILE           *fp;
    shaStrings     *pShaStrings;
{
    int             i;

    fprintf(fp, "%d\n", pShaStrings->shaStrings_len);
    for (i = 0; i < pShaStrings->shaStrings_len; i++) {
        fprintf(fp, "%s\n", pShaStrings->shaStrings_val[i]);
    }

}

void
shaStringsXDRFree(pShaStrings)
    shaStrings     *pShaStrings;
{
    xdr_free(xdr_shaStrings, (char *) pShaStrings);
    memset(pShaStrings,0,sizeof(shaStrings));
}

int
shaBunchOut(fd, pShaBunch)
    int             fd;
    shaBunch      *pShaBunch;
{
    if(!xdr_shaBunch(mplexXDRSEnc(fd), pShaBunch)){
        return -1;
    }
    return 1;
}

int
shaBunchIn(fd, pShaBunch)
    int             fd;
    shaBunch      *pShaBunch;
{
    shaBunchXDRFree(pShaBunch);
    if(!xdr_shaBunch(mplexXDRSDec(fd), pShaBunch)){
```

```
        return -1;
    }
    return 1;
}

int
shaBunchsOut(fd, pShaBunchs)
    int             fd;
    shaBunchs     *pShaBunchs;
{
    if(!xdr_shaBunchs(mplexXDRSEnc(fd), pShaBunchs)){
        return  -1;
    }
    return 1;
}

int
shaBunchsIn(fd, pShaBunchs)
    int             fd;
    shaBunchs     *pShaBunchs;
{
    shaBunchsXDRFree(pShaBunchs);
    if(!xdr_shaBunchs(mplexXDRSDec(fd), pShaBunchs)){
        return -1;
    }
    return 1;
}

void
freeShaBunch(pShaBunch)
    shaBunch     *pShaBunch;
{
    int             i;

    if (pShaBunch == NULL) {
        return;
    }
    free(pShaBunch->shaBunch_val);
    memset(pShaBunch,0, sizeof(shaBunch));
}


void
freeShaBunchs(pShaBunchs)
    shaBunchs     *pShaBunchs;
{
    int             i;

    if (pShaBunchs == NULL) {
        return;
    }
    for(i=0;i<pShaBunchs->shaBunchs_len;i++){
        freeShaBunch(&pShaBunchs->shaBunchs_val[i]);
```

```
    }
    free(pShaBunchs->shaBunchs_val);
    memset(pShaBunchs,0,sizeof(shaBunchs));
}

shaBunch     *
copyShaBunch(pShaBunch, destpShaBunch)
    shaBunch     *pShaBunch;
    shaBunch     *destpShaBunch;
{
    int             i;
    shaBunch     *newpShaBunch;

    if (pShaBunch == NULL) {
        return NULL;
    }
    if (destpShaBunch == NULL) {
        newpShaBunch = (shaBunch *) malloc(sizeof(shaBunch));
    } else {
        newpShaBunch = destpShaBunch;
    }
    memcpy(newpShaBunch, pShaBunch, sizeof(shaBunch));
    newpShaBunch->shaBunch_val = (char *)
        malloc(newpShaBunch->shaBunch_len);
    memcpy(newpShaBunch->shaBunch_val, pShaBunch->shaBunch_val,
        pShaBunch->shaBunch_len);
    return newpShaBunch;
}
shaBunchs     *
copyShaBunchs(pShaBunchs, destpShaBunchs)
    shaBunchs     *pShaBunchs;
    shaBunchs     *destpShaBunchs;
{
    int             i;
    shaBunchs     *newpShaBunchs;

    if (pShaBunchs == NULL) {
        return NULL;
    }
    if (destpShaBunchs == NULL) {
        newpShaBunchs = (shaBunchs *) malloc(sizeof(shaBunchs));
    } else {
        newpShaBunchs = destpShaBunchs;
    }
    memcpy(newpShaBunchs, pShaBunchs, sizeof(shaBunchs));
    newpShaBunchs->shaBunchs_val = (shaBunch *)
        malloc(newpShaBunchs->shaBunchs_len * sizeof(shaBunch));
    for(i=0; i<pShaBunchs->shaBunchs_len;i++){
        copyShaBunch(&pShaBunchs->shaBunchs_val[i],
            &newpShaBunchs->shaBunchs_val[i]);
    }
    return newpShaBunchs;
}
```

```
void
inputShaBunch(fp, pShaBunch)
    FILE           *fp;
    shaBunch     *pShaBunch;
{
    int             i;

    fscanf(fp, "%d", &pShaBunch->shaBunch_len);
    pShaBunch->shaBunch_val = (char *)
        malloc(pShaBunch->shaBunch_len );
    for (i = 0; i < pShaBunch->shaBunch_len; i++) {
        pShaBunch->shaBunch_val[i] = fgetc(fp);
    }
}

void
outputShaBunch(fp, pShaBunch)
    FILE           *fp;
    shaBunch     *pShaBunch;
{
    int             i;

    fprintf(fp, "%d\n", pShaBunch->shaBunch_len);
    for (i = 0; i < pShaBunch->shaBunch_len; i++) {
        fputc(pShaBunch->shaBunch_val[i], fp);
    }

}

void
inputShaBunchs(fp, pShaBunchs)
    FILE           *fp;
    shaBunchs     *pShaBunchs;
{
    int             i;

    fscanf(fp, "%d", &pShaBunchs->shaBunchs_len);
    pShaBunchs->shaBunchs_val = (shaBunch *)
        malloc(pShaBunchs->shaBunchs_len * sizeof(shaBunch));
    for (i = 0; i < pShaBunchs->shaBunchs_len; i++) {
        inputShaBunch(fp,&pShaBunchs->shaBunchs_val[i]);
    }
}

void
outputShaBunchs(fp, pShaBunchs)
    FILE           *fp;
    shaBunchs     *pShaBunchs;
{
    int             i;

    fprintf(fp, "%d\n", pShaBunchs->shaBunchs_len);
```

```
    for (i = 0; i < pShaBunchs->shaBunchs_len; i++) {
        outputShaBunch(fp, &pShaBunchs->shaBunchs_val[i]);
    }

}

void
shaBunchXDRFree(pShaBunch)
    shaBunch     *pShaBunch;
{
    xdr_free(xdr_shaBunch, (char *) pShaBunch);
    memset(pShaBunch,0,sizeof(shaBunch));
}

void
shaBunchsXDRFree(pShaBunchs)
    shaBunchs     *pShaBunchs;
{
    xdr_free(xdr_shaBunchs, (char *) pShaBunchs);
    memset(pShaBunchs,0,sizeof(shaBunchs));
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/shastraData.h>

bool_t
xdr_shaObjId(xdrs, objp)
    XDR *xdrs;
    shaObjId *objp;
{
    if (!xdr_u_long(xdrs, &objp->lSIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lIdTag)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaObjId_P(xdrs, objp)
    XDR *xdrs;
    shaObjId_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaObjId), xdr_shaObjId))
        {
```

```
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaObjIds(xdrs, objp)
    XDR *xdrs;
    shaObjIds *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaObjIds_val, (u_int *)&objp->
        shaObjIds_len, ~0, sizeof(shaObjId), xdr_shaObjId)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaObjIds_P(xdrs, objp)
    XDR *xdrs;
    shaObjIds_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaObjIds), xdr_shaObjIds)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaChar(xdrs, objp)
    XDR *xdrs;
    shaChar *objp;
{
    if (!xdr_char(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaChars(xdrs, objp)
    XDR *xdrs;
    shaChars *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaChars_val, (u_int *)&objp->
        shaChars_len, ~0, sizeof(char), xdr_char)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
```

```
xdr_shaChars_P(xdrs, objp)
    XDR *xdrs;
    shaChars_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaChars), xdr_shaChars))
        {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUChar(xdrs, objp)
    XDR *xdrs;
    shaUChar *objp;
{
    if (!xdr_u_char(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUChars(xdrs, objp)
    XDR *xdrs;
    shaUChars *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaUChars_val, (u_int *)&objp->
        shaUChars_len, ~0, sizeof(u_char), xdr_u_char)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUChars_P(xdrs, objp)
    XDR *xdrs;
    shaUChars_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaUChars), xdr_shaUChars)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaShort(xdrs, objp)
    XDR *xdrs;
    shaShort *objp;
{
    if (!xdr_short(xdrs, objp)) {
        return (FALSE);
```

```
    }
    return (TRUE);
}

bool_t
xdr_shaShorts(xdrs, objp)
    XDR *xdrs;
    shaShorts *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaShorts_val, (u_int *)&objp->
        shaShorts_len, ~0, sizeof(short), xdr_short)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaShorts_P(xdrs, objp)
    XDR *xdrs;
    shaShorts_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaShorts), xdr_shaShorts)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUShort(xdrs, objp)
    XDR *xdrs;
    shaUShort *objp;
{
    if (!xdr_u_short(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUShorts(xdrs, objp)
    XDR *xdrs;
    shaUShorts *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaUShorts_val, (u_int *)&objp->
        shaUShorts_len, ~0, sizeof(u_short), xdr_u_short)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUShorts_P(xdrs, objp)
```

```
    XDR *xdrs;
    shaUShorts_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaUShorts),
        xdr_shaUShorts)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaInt(xdrs, objp)
    XDR *xdrs;
    shaInt *objp;
{
    if (!xdr_int(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaInts(xdrs, objp)
    XDR *xdrs;
    shaInts *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaInts_val, (u_int *)&objp->
        shaInts_len, ~0, sizeof(int), xdr_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaInts_P(xdrs, objp)
    XDR *xdrs;
    shaInts_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaInts), xdr_shaInts)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUInt(xdrs, objp)
    XDR *xdrs;
    shaUInt *objp;
{
    if (!xdr_u_int(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
```

```
}

bool_t
xdr_shaUInts(xdrs, objp)
    XDR *xdrs;
    shaUInts *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaUInts_val, (u_int *)&objp->
        shaUInts_len, ~0, sizeof(u_int), xdr_u_int)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaUInts_P(xdrs, objp)
    XDR *xdrs;
    shaUInts_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaUInts), xdr_shaUInts))
        {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaLong(xdrs, objp)
    XDR *xdrs;
    shaLong *objp;
{
    if (!xdr_long(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaLongs(xdrs, objp)
    XDR *xdrs;
    shaLongs *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaLongs_val, (u_int *)&objp->
        shaLongs_len, ~0, sizeof(long), xdr_long)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaLongs_P(xdrs, objp)
    XDR *xdrs;
    shaLongs_P *objp;
```

```
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaLongs), xdr_shaLongs))
        {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaULong(xdrs, objp)
    XDR *xdrs;
    shaULong *objp;
{
    if (!xdr_u_long(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaULongs(xdrs, objp)
    XDR *xdrs;
    shaULongs *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaULongs_val, (u_int *)&objp->
        shaULongs_len, ~0, sizeof(u_long), xdr_u_long)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaULongs_P(xdrs, objp)
    XDR *xdrs;
    shaULongs_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaULongs), xdr_shaULongs)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaFloat(xdrs, objp)
    XDR *xdrs;
    shaFloat *objp;
{
    if (!xdr_float(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
bool_t
xdr_shaFloats(xdrs, objp)
    XDR *xdrs;
    shaFloats *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaFloats_val, (u_int *)&objp->
        shaFloats_len, ~0, sizeof(float), xdr_float)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaFloats_P(xdrs, objp)
    XDR *xdrs;
    shaFloats_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaFloats), xdr_shaFloats)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaDouble(xdrs, objp)
    XDR *xdrs;
    shaDouble *objp;
{
    if (!xdr_double(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaDoubles(xdrs, objp)
    XDR *xdrs;
    shaDoubles *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaDoubles_val, (u_int *)&objp->
        shaDoubles_len, ~0, sizeof(double), xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaDoubles_P(xdrs, objp)
    XDR *xdrs;
    shaDoubles_P *objp;
{
```

```
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaDoubles),
        xdr_shaDoubles)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaString(xdrs, objp)
    XDR *xdrs;
    shaString *objp;
{
    if (!xdr_string(xdrs, objp, ~0)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaStrings(xdrs, objp)
    XDR *xdrs;
    shaStrings *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaStrings_val, (u_int *)&objp->
        shaStrings_len, ~0, sizeof(shaString), xdr_shaString)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaStrings_P(xdrs, objp)
    XDR *xdrs;
    shaStrings_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaStrings),
        xdr_shaStrings)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaBunch(xdrs, objp)
    XDR *xdrs;
    shaBunch *objp;
{
    if (!xdr_bytes(xdrs, (char **)&objp->shaBunch_val, (u_int *)&objp->
        shaBunch_len, ~0)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
bool_t
xdr_shaBunchs(xdrs, objp)
    XDR *xdrs;
    shaBunchs *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shaBunchs_val, (u_int *)&objp->
        shaBunchs_len, ~0, sizeof(shaBunch), xdr_shaBunch)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shaBunchs_P(xdrs, objp)
    XDR *xdrs;
    shaBunchs_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shaBunchs), xdr_shaBunchs)
        ) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>

#include <shastra/datacomm/shastraHdrH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

#define STANDALONEnn

int
shastraHdrOut(fd, pSHdr)
    int             fd;
    shastraHdr      *pSHdr;
{
    XDR             xdrs;
    int     retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if(!xdr_shastraHdr(&xdrs, pSHdr)){
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
    xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
    */
```

```
    if(!xdr_shastraHdr(mplexXDRSEnc(fd), pSHdr)){
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraHdrIn(fd, pSHdr)
    int             fd;
    shastraHdr      *pSHdr;
{
    XDR             xdrs;
    int     retVal = 0;

    shastraHdrXDRFree(pSHdr);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if(!xdr_shastraHdr(&xdrs, pSHdr)){
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
    xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
    */
    if(!xdr_shastraHdr(mplexXDRSDec(fd), pSHdr)){
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}


void
inputHdr(fp, pSHdr)
    FILE           *fp;
    shastraHdr      *pSHdr;
{
    fscanf(fp, "%c", &pSHdr->bProtocol);
    fscanf(fp, "%c", &pSHdr->bVersion);
    fscanf(fp, "%lu", &pSHdr->lSize);
    fscanf(fp, "%lu", &pSHdr->lMesgId);
    fscanf(fp, "%lu", &pSHdr->senderTag);
    fscanf(fp, "%lu", &pSHdr->recvrTag);
    fscanf(fp, "%lu", &pSHdr->lOpCode);
    fscanf(fp, "%hu", &pSHdr->sMsgType);
    fscanf(fp, "%hu", &pSHdr->sNumMsgs);
    fscanf(fp, "%hu", &pSHdr->sSeqNum);
```

```
}

void
outputHdr(fp, pSHdr)
    FILE           *fp;
    shastraHdr      *pSHdr;
{
    fprintf(fp, "%hu\n", &pSHdr->bProtocol);
    fprintf(fp, "%hu\n", &pSHdr->bVersion);
    fprintf(fp, "%lu\n", &pSHdr->lSize);
    fprintf(fp, "%lu\n", &pSHdr->lMesgId);
    fprintf(fp, "%lu\n", &pSHdr->senderTag);
    fprintf(fp, "%lu\n", &pSHdr->recvrTag);
    fprintf(fp, "%lu\n", &pSHdr->lOpCode);
    fprintf(fp, "%hu\n", &pSHdr->sMsgType);
    fprintf(fp, "%hu\n", &pSHdr->sNumMsgs);
    fprintf(fp, "%hu\n", &pSHdr->sSeqNum);

}


void
freeHdr(pSHdr)
    shastraHdr      *pSHdr;
{
    int             i;

    if (pSHdr == NULL) {
        return;
    }
    memset(pSHdr,0, sizeof(shastraHdr));
}


shastraHdr      *
copyHdr(pSHdr, destpSHdr)
    shastraHdr      *pSHdr;
    shastraHdr      *destpSHdr;
{
    shastraHdr      *newpSHdr;
    int             i;

    if (pSHdr == NULL) {
        return NULL;
    }
    if (destpSHdr == NULL) {
        newpSHdr = (shastraHdr *) malloc(sizeof(shastraHdr));
    } else {
        newpSHdr = destpSHdr;
    }

    memcpy(newpSHdr, pSHdr, sizeof(shastraHdr));
    return newpSHdr;
```

```
}

#ifdef STANDALONE
main(argc, argv)
#else                /* STANDALONE */
shastraHdrMain(argc, argv)
#endif               /* STANDALONE */
    int              argc;
    char           **argv;
{
    static shastraHdr sHdr;
    shastraHdr      *cpSHdr;

    switch (argc) {
    case 1:     /* receive sHdr */
        shastraHdrIn(0 /* stdin */ , &sHdr);
        outputHdr(stdout, &sHdr);
        cpSHdr = copyHdr(&sHdr, NULL);
        outputHdr(stdout, cpSHdr);
        freeHdr(cpSHdr);

        break;
    case 2:     /* receive sHdr */
        inputHdr(stdin, &sHdr);
#ifdef DEBUG
        outputHdr(stderr, &sHdr);
#endif
        shastraHdrOut(1 /* stdout */ , &sHdr);

        break;
    }

}


void
shastraHdrXDRFree(pSHdr)
    shastraHdr      *pSHdr;
{
    xdr_free(xdr_shastraHdr, (char *) pSHdr);
    memset(pSHdr, 0, sizeof(shastraHdr));
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/shastraHdr.h>

bool_t
xdr_shastraHdr(xdrs, objp)
    XDR *xdrs;
    shastraHdr *objp;
{
    if (!xdr_u_char(xdrs, &objp->bProtocol)) {
        return (FALSE);
    }
    if (!xdr_u_char(xdrs, &objp->bVersion)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lSize)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lMesgId)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->senderTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->recvrTag)) {
```

```
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lOpCode)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->sMsgType)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->sNumMsgs)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->sSeqNum)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shastraHdr_P(xdrs, objp)
    XDR *xdrs;
    shastraHdr_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shastraHdr),
        xdr_shastraHdr)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <string.h>

#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>
#include <shastra/shastra.h>
#include <shastra/datacomm/shastraIdH.h>

#define STANDALONEnn

int
shastraIdOut(fd, pSId)
    int             fd;
    shastraId      *pSId;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_shastraId(&xdrs, pSId)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
```

```
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_shastraId(mplexXDRSEnc(fd), pSId)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdIn(fd, pSId)
    int             fd;
    shastraId      *pSId;
{
    XDR             xdrs;
    int             retVal = 0;

    shastraIdXDRFree(pSId);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_shastraId(&xdrs, pSId)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_shastraId(mplexXDRSDec(fd), pSId)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
#ifdef SHASTRA4IRIX6
     {
        int temp;
        temp = pSId->lIPAddr;
        pSId->lIPAddr = 0;
        pSId->lIPAddr = temp;
        pSId->lIPAddr = pSId->lIPAddr & 0x00000000ffffffff;
     }
#endif
    return retVal;
}

int
shastraIdMemOut(buf, size, pSId)
    char           *buf;
    int             size;
    shastraId       *pSId;
{
```

```
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_shastraId(&xdrs, pSId)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
shastraIdMemIn(buf, size, pSId)
    char           *buf;
    int             size;
    shastraId       *pSId;
{
    XDR             xdrs;
    int             retVal = 0;

    shastraIdXDRFree(pSId);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_shastraId(&xdrs, pSId)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
shastraIdsOut(fd, pSIds)
    int             fd;
    shastraIds     *pSIds;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_shastraIds(&xdrs, pSIds)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_shastraIds(mplexXDRSEnc(fd), pSIds)) {
        retVal = -1;
    }
```

```
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdsIn(fd, pSIds)
    int             fd;
    shastraIds     *pSIds;
{
    XDR             xdrs;
    int             retVal = 0;

    shastraIdsXDRFree(pSIds);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_shastraIds(&xdrs, pSIds)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_shastraIds(mplexXDRSDec(fd), pSIds)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdsMemOut(buf, size, pSIds)
    char           *buf;
    int             size;
    shastraIds       *pSIds;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_shastraIds(&xdrs, pSIds)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
shastraIdsMemIn(buf, size, pSIds)
    char           *buf;
```

```
    int             size;
    shastraIds       *pSIds;
{
    XDR             xdrs;
    int             retVal = 0;

    shastraIdsXDRFree(pSIds);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_shastraIds(&xdrs, pSIds)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}


void
inputId(fp, pSId)
    FILE           *fp;
    shastraId      *pSId;
{
    int             i;
    char sbBuf[128];

    fscanf(fp, "%s", sbBuf);
    pSId->nmHost = strdup(sbBuf);
    fscanf(fp, "%s", sbBuf);
    pSId->nmDisplay = strdup(sbBuf);
    fscanf(fp, "%s", sbBuf);
    pSId->nmApplicn = strdup(sbBuf);
    fscanf(fp, "%s", sbBuf);
    pSId->nmUser = strdup(sbBuf);
    fscanf(fp, "%s", sbBuf);
    pSId->nmPasswd = strdup(sbBuf);
    fscanf(fp, "%l", &pSId->lIPAddr);
    fscanf(fp, "%l", &pSId->lSIDTag);
    fscanf(fp, "%d", &pSId->iPort);
    fscanf(fp, "%d", &pSId->iProcId);
    fscanf(fp, "%d", &pSId->lPerms);
    fscanf(fp, "%d", &pSId->lHWState);
    fscanf(fp, "%lf", &pSId->dLoadAvg);
    pSId->iProcId = getpid();
}

void
outputId(fp, pSId)
    FILE           *fp;
    shastraId      *pSId;
{
    int             i;

    fprintf(fp, "%s\n", pSId->nmHost);
    fprintf(fp, "%s\n", pSId->nmDisplay);
```

```
    fprintf(fp, "%s\n", pSId->nmApplicn);
    fprintf(fp, "%s\n", pSId->nmUser);
    fprintf(fp, "%s\n", pSId->nmPasswd);
    fprintf(fp, "%s\n", ipaddr2str(pSId->lIPAddr));
    fprintf(fp, "%lu\n", pSId->lSIDTag);
    fprintf(fp, "%d\n", pSId->iPort);
    fprintf(fp, "%d\n", pSId->iProcId);
    fprintf(fp, "%d\n", pSId->lPerms);
    fprintf(fp, "%d\n", pSId->lHWState);
    fprintf(fp, "%lf\n", pSId->dLoadAvg);

}


void
inputIds(fp, pSIds)
    FILE           *fp;
    shastraIds     *pSIds;
{
    int             i;

    fscanf(fp, "%d", &pSIds->shastraIds_len);
    pSIds->shastraIds_val = (shastraId_P *)
        malloc(pSIds->shastraIds_len * sizeof(shastraId_P));
    for (i = 0; i < pSIds->shastraIds_len; i++) {
        pSIds->shastraIds_val[i] = (shastraId_P) malloc(sizeof(shastraId));
        inputId(fp, pSIds->shastraIds_val[i]);
    }
}

void
outputIds(fp, pSIds)
    FILE           *fp;
    shastraIds     *pSIds;
{
    int             i;

    fprintf(fp, "%d\n", pSIds->shastraIds_len);
    for (i = 0; i < pSIds->shastraIds_len; i++) {
        outputId(fp, pSIds->shastraIds_val[i]);
    }

}


void
freeId(pSId)
    shastraId      *pSId;
{
    int             i;

    if (pSId == NULL) {
        return;
```

```
    }
    free(pSId->nmHost);
    free(pSId->nmDisplay);
    free(pSId->nmApplicn);
    free(pSId->nmUser);
    memset(pSId, 0, sizeof(shastraId));
}


void
freeIds(pSIds)
    shastraIds     *pSIds;
{
    int             i;

    if (pSIds == NULL) {
        return;
    }
    for (i = 0; i < pSIds->shastraIds_len; i++) {
        freeId(pSIds->shastraIds_val[i]);
    }
    free(pSIds->shastraIds_val);
    memset(pSIds, 0, sizeof(shastraIds));
}

shastraId      *
copyId(pSId, destpSId)
    shastraId      *pSId;
    shastraId      *destpSId;
{
    shastraId      *newpSId;
    int             i;

    if (pSId == NULL) {
        return NULL;
    }
    if (destpSId == NULL) {
        newpSId = (shastraId *) malloc(sizeof(shastraId));
    } else {
        newpSId = destpSId;
    }

    memcpy(newpSId, pSId, sizeof(shastraId));
    if(pSId->nmHost){
        newpSId->nmHost = strdup(pSId->nmHost);
    }
    if(pSId->nmDisplay){
        newpSId->nmDisplay = strdup(pSId->nmDisplay);
    }
    if(pSId->nmApplicn){
        newpSId->nmApplicn = strdup(pSId->nmApplicn);
    }
    if(pSId->nmUser){
```

```
        newpSId->nmUser = strdup(pSId->nmUser);
    }
    if(pSId->nmPasswd){
        newpSId->nmPasswd = strdup(pSId->nmPasswd);
    }
    return newpSId;
}

shastraIds     *
copyIds(pSIds, destpSIds)
    shastraIds     *pSIds;
    shastraIds     *destpSIds;
{
    int             i;
    shastraIds     *newpSIds;

    if (pSIds == NULL) {
        return NULL;
    }
    if (destpSIds == NULL) {
        newpSIds = (shastraIds *) malloc(sizeof(shastraIds));
    } else {
        newpSIds = destpSIds;
    }
    memcpy(newpSIds, pSIds, sizeof(shastraIds));
    newpSIds->shastraIds_val = (shastraId_P *)
        malloc(newpSIds->shastraIds_len * sizeof(shastraId_P));
    for (i = 0; i < newpSIds->shastraIds_len; i++) {
        newpSIds->shastraIds_val[i] = copyId(pSIds->shastraIds_val[i], NULL
            );
    }
    return newpSIds;
}


#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
shastraIdMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
    static shastraId sId;
    static shastraIds sIds;
    shastraIds     *cpSIds;
    shastraId      *cpSId;

    switch (argc) {
    case 1:     /* receive sId */
        shastraIdIn(0 /* stdin */ , &sId);
        outputId(stdout, &sId);
        cpSId = copyId(&sId, NULL);
```

```
        outputId(stdout, cpSId);
        freeId(cpSId);

        break;
    case 2:     /* receive sId */
        inputId(stdin, &sId);
#ifdef DEBUG
        outputId(stderr, &sId);
#endif
        shastraIdOut(1 /* stdout */ , &sId);

        break;
    case 3:     /* receive sIds */
        shastraIdsIn(0 /* stdin */ , &sIds);
        outputIds(stdout, &sIds);
        cpSIds = copyIds(&sIds, NULL);
        outputIds(stdout, cpSIds);
        freeIds(cpSIds);

        break;
    case 4:     /* receive sIds */
        inputIds(stdin, &sIds);
#ifdef DEBUG
        outputIds(stderr, &sIds);
#endif
        shastraIdsOut(1 /* stdout */ , &sIds);

        break;
    }

}

/*
 * Function --
 */
char           *
ipaddr2str(addr)
    unsigned long   addr;
{
    static char     addrBuf[32];
    unsigned int    b1, b2, b3, b4;

    b4 = addr % 256;
    addr = addr / 256;
    b3 = addr % 256;
    addr = addr / 256;
    b2 = addr % 256;
    addr = addr / 256;
    b1 = addr % 256;
    sprintf(addrBuf, "%d.%d.%d.%d", b1, b2, b3, b4);

    return addrBuf;
}
```

```
/*
 * Function --
 */
char           *
pSId2Str(pSId, fMask)
    shastraId      *pSId;
    int             fMask;
{
    char           *buf, *bufptr, tmpHost[256], *tmp;
    int             i;
    int StrMaxLen = 128;

    /* shastraId has 9 displayable fields 4+n names, rest num */
    buf = malloc((StrMaxLen + 1) * 6);
    bufptr = buf;

    if ((fMask == 0) || (fMask == PSIDSHOWALL)) {
        strcpy(tmpHost, pSId->nmHost);
        tmp = strchr(tmpHost, '.');
        if(tmp){
            *tmp = '\0';
        }
#ifndef  VERBOSE
if (pSId->webname)
{
    sprintf(buf, "%s@%s's %s (Tag: %u)\n",
    pSId->webname, tmpHost, pSId->nmApplicn, pSId->lSIDTag);
}
else
{
    sprintf(buf, "%s@%s's %s (Tag: %u)\n",
    pSId->nmUser, tmpHost, pSId->nmApplicn, pSId->lSIDTag);
}
#else                /* VERBOSE */
        sprintf(buf, "%s@%s's %s (pid %d, ip %s, port %d) on display %s\n",
            pSId->nmUser, tmpHost, pSId->nmApplicn, pSId->iProcId,
           ipaddr2str(pSId->lIPAddr), pSId->iPort, pSId->nmDisplay);
#endif               /* VERBOSE */

    } else {
        if (fMask & PSIDNMUSER) {
            sprintf(bufptr, "%s ", pSId->nmUser);
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDNMHOST) {
            if(fMask == PSIDNMHOST){
                sprintf(bufptr, "%s ", pSId->nmHost);
            }
            else{
                strcpy(tmpHost, pSId->nmHost);
                tmp = strchr(tmpHost, '.');
                if(tmp){
```

```
                    *tmp = '\0';
                }
                sprintf(bufptr, "%s ", tmpHost);
            }
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDNMAPPL) {
            sprintf(bufptr, "%s ", pSId->nmApplicn);
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDNMDISP) {
            sprintf(bufptr, "%s ", pSId->nmDisplay);
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDIPADDR) {
            sprintf(bufptr, "%s ", ipaddr2str(pSId->lIPAddr));
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDPORT) {
            sprintf(bufptr, "%d ", pSId->iPort);
            bufptr += strlen(bufptr);
        }
        if (fMask & PSIDPROCID) {
            sprintf(bufptr, "%d ", pSId->iProcId);
            bufptr += strlen(bufptr);
        }
        if (pSId->lPerms && (fMask & PSIDPERMS)) {
            char           *tmp;
            tmp = perms2Str(pSId->lPerms);
            sprintf(bufptr, "%s ", tmp);
            free(tmp);
            bufptr += strlen(bufptr);
        }
    }

    buf = realloc(buf, strlen(buf) + 1);
    return buf;
}

/*
 * Function --
 */
char           *
perms2Str(perms)
    unsigned long   perms;
{
    char           *buf;

    buf = malloc(16);
    sprintf(buf, "(%c%c%c%c%c) ",
        (perms & SHASTRA_PERM_ACCESS) ? 'A' : ' ',
        (perms & SHASTRA_PERM_BROWSE) ? 'B' : ' ',
        (perms & SHASTRA_PERM_MODIFY) ? 'M' : ' ',
```

```
        (perms & SHASTRA_PERM_COPY) ? 'C' : ' ',
        (perms & SHASTRA_PERM_GRANT) ? 'G' : ' ');
    return buf;
}
/*
 * Function --
 */
char          **
pSIds2StrTab(pSIds, fMask)
    shastraIds     *pSIds;
    int             fMask;
{
    int             i;
    char          **buf;

    buf = (char **) malloc(sizeof(char *) * (pSIds->shastraIds_len + 1));
    for (i = 0; i < pSIds->shastraIds_len; i++) {
        buf[i] = pSId2Str(pSIds->shastraIds_val[i], fMask);
    }
    buf[pSIds->shastraIds_len] = NULL;
    return buf;
}

/*
 * Function
 */
char *
pSId2StrDetail(pSId, lPerms)
    shastraId      *pSId;
    unsigned long lPerms;
{
    char           *sb, *sbBuf;

    sbBuf = malloc(1024);

    sb = sbBuf;
    sprintf(sb, "Application    : %s\n", pSId->nmApplicn);
    sb += strlen(sb);
    sprintf(sb, "User Name      : %s\n", pSId->nmUser);
    sb += strlen(sb);
    sprintf(sb, "X Display      : %s\n", pSId->nmDisplay);
    sb += strlen(sb);
    sprintf(sb, "Host Name      : %s\n", pSId->nmHost);
    sb += strlen(sb);
    sprintf(sb, "IP Address     : %s\n", ipaddr2str(pSId->lIPAddr));
    sb += strlen(sb);
    sprintf(sb, "Load Average   : %lf\n", pSId->dLoadAvg);
    sb += strlen(sb);
    sprintf(sb, "Host ID Tag    : %lu\n", pSId->lSIDTag);
    sb += strlen(sb);
    sprintf(sb, "TCP Port       : %d\n", pSId->iPort);
    sb += strlen(sb);
    sprintf(sb, "Process ID     : %d\n", pSId->iProcId);
```

```
    sb += strlen(sb);
    if (lPerms) {
        sprintf(sb, "Permissions    : (%c%c%c%c%c)\n",
            (pSId->lPerms & SHASTRA_PERM_ACCESS) ? 'A' : ' ',
            (pSId->lPerms & SHASTRA_PERM_BROWSE) ? 'B' : ' ',
            (pSId->lPerms & SHASTRA_PERM_MODIFY) ? 'M' : ' ',
            (pSId->lPerms & SHASTRA_PERM_COPY) ? 'C' : ' ',
            (pSId->lPerms & SHASTRA_PERM_GRANT) ? 'G' : ' ');
        sb += strlen(sb);
    }
    sprintf(sb, "\n");

    return sbBuf;
}

void
shastraIdXDRFree(pSId)
    shastraId      *pSId;
{
    xdr_free(xdr_shastraId, (char *) pSId);
    memset(pSId, 0, sizeof(shastraId));
}

void
shastraIdsXDRFree(pSIds)
    shastraIds     *pSIds;
{
    xdr_free(xdr_shastraIds, (char *) pSIds);
    memset(pSIds, 0, sizeof(shastraIds));
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/shastraIdTag.h>
#include <shastra/datacomm/shastraId.h>

bool_t
xdr_shastraId(xdrs, objp)
    XDR *xdrs;
    shastraId *objp;
{
    if (!xdr_string(xdrs, &objp->nmHost, ~0)) {
        return (FALSE);
    }
    if (!xdr_string(xdrs, &objp->nmDisplay, ~0)) {
        return (FALSE);
    }
    if (!xdr_string(xdrs, &objp->nmApplicn, ~0)) {
        return (FALSE);
    }
    if (!xdr_string(xdrs, &objp->nmUser, ~0)) {
        return (FALSE);
    }
    if (!xdr_string(xdrs, &objp->nmPasswd, ~0)) {
        return (FALSE);
    }
```

```
    if (!xdr_shastraIdTag(xdrs, &objp->lSIDTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lIPAddr)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lWindowId)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lApplicn)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->iPort)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->iProcId)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->iXPort)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->iXScreen)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lPerms)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lHWState)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lTimeStamp)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->dLoadAvg)) {
        return (FALSE);
    }
    if (!xdr_string(xdrs, &objp->webname, ~0)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shastraId_P(xdrs, objp)
    XDR *xdrs;
    shastraId_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shastraId), xdr_shastraId)
        ) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
bool_t
xdr_shastraIds(xdrs, objp)
    XDR *xdrs;
    shastraIds *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shastraIds_val, (u_int *)&objp->
        shastraIds_len, ~0, sizeof(shastraId_P), xdr_shastraId_P)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shastraIdGrp(xdrs, objp)
    XDR *xdrs;
    shastraIdGrp *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shastraIdGrp_val, (u_int *)&objp->
        shastraIdGrp_len, ~0, sizeof(shastraId), xdr_shastraId)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <malloc.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

#define STANDALONEnn

int
shastraIdTagOut(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_shastraIdTag(&xdrs, pSIdTag)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
```

```
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_shastraIdTag(mplexXDRSEnc(fd), pSIdTag)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdTagIn(fd, pSIdTag)
    int             fd;
    shastraIdTag   *pSIdTag;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_shastraIdTag(&xdrs, pSIdTag)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_shastraIdTag(mplexXDRSDec(fd), pSIdTag)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdTagsOut(fd, pSIdTags)
    int             fd;
    shastraIdTags  *pSIdTags;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_shastraIdTags(&xdrs, pSIdTags)) {
            retVal = -1;
        }
```

```
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_shastraIdTags(mplexXDRSEnc(fd), pSIdTags)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
shastraIdTagsIn(fd, pSIdTags)
    int             fd;
    shastraIdTags  *pSIdTags;
{
    XDR             xdrs;
    int             retVal = 0;

    shastraIdTagsXDRFree(pSIdTags);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_shastraIdTags(&xdrs, pSIdTags)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_shastraIdTags(mplexXDRSDec(fd), pSIdTags)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}


void
inputIdTag(fp, pSIdTag)
    FILE           *fp;
    shastraIdTag   *pSIdTag;
{
    int             i;

    fscanf(fp, "%lu", pSIdTag);
}

void
```

```
outputIdTag(fp, pSIdTag)
    FILE           *fp;
    shastraIdTag   *pSIdTag;
{
    int             i;

    fprintf(fp, "%lu\n", *pSIdTag);
}


void
inputIdTags(fp, pSIdTags)
    FILE           *fp;
    shastraIdTags  *pSIdTags;
{
    int             i;

    fscanf(fp, "%u", &pSIdTags->shastraIdTags_len);
    pSIdTags->shastraIdTags_val = (shastraIdTag *)
        malloc(pSIdTags->shastraIdTags_len * sizeof(shastraIdTag));
    for (i = 0; i < pSIdTags->shastraIdTags_len; i++) {
        inputIdTag(fp, &pSIdTags->shastraIdTags_val[i]);
    }
}

void
outputIdTags(fp, pSIdTags)
    FILE           *fp;
    shastraIdTags  *pSIdTags;
{
    int             i;

    fprintf(fp, "%u\n", pSIdTags->shastraIdTags_len);
    for (i = 0; i < pSIdTags->shastraIdTags_len; i++) {
        outputIdTag(fp, &pSIdTags->shastraIdTags_val[i]);
    }

}


void
freeIdTags(pSIdTags)
    shastraIdTags  *pSIdTags;
{
    int             i;

    if (pSIdTags == NULL) {
        return;
    }
    free(pSIdTags->shastraIdTags_val);
    memset(pSIdTags, 0, sizeof(shastraIdTags));
}
```

```
shastraIdTags  *
copyIdTags(pSIdTags, destpSIdTags)
    shastraIdTags  *pSIdTags;
    shastraIdTags  *destpSIdTags;
{
    int             i;
    shastraIdTags  *newpSIdTags;

    if (pSIdTags == NULL) {
        return NULL;
    }
    if (destpSIdTags == NULL) {
        newpSIdTags = (shastraIdTags *) malloc(sizeof(shastraIdTags));
    } else {
        newpSIdTags = destpSIdTags;
    }
    memcpy(newpSIdTags, pSIdTags, sizeof(shastraIdTags));
    newpSIdTags->shastraIdTags_val = (shastraIdTag *)
        malloc(newpSIdTags->shastraIdTags_len * sizeof(shastraIdTag));
    for (i = 0; i < newpSIdTags->shastraIdTags_len; i++) {
        newpSIdTags->shastraIdTags_val[i] =
            pSIdTags->shastraIdTags_val[i];
    }
    return newpSIdTags;
}


#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
shastraIdTagMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
    static shastraIdTag sIdTag;
    static shastraIdTags sIdTags;
    shastraIdTags  *cpSIdTags;
    shastraIdTag    cpSIdTag;

    switch (argc) {
    case 1:     /* receive sId */
        shastraIdTagIn(0 /* stdin */ , &sIdTag);
        outputIdTag(stdout, &sIdTag);
        cpSIdTag = sIdTag;
        outputIdTag(stdout, &cpSIdTag);

        break;
    case 2:     /* receive sId */
        inputIdTag(stdin, &sIdTag);
#ifdef DEBUG
        outputIdTag(stderr, &sIdTag);
```

```
#endif
        shastraIdTagOut(1 /* stdout */ , &sIdTag);

        break;
    case 3:     /* receive sIds */
        shastraIdTagsIn(0 /* stdin */ , &sIdTags);
        outputIdTags(stdout, &sIdTags);
        cpSIdTags = copyIdTags(&sIdTags, NULL);
        outputIdTags(stdout, cpSIdTags);
        freeIdTags(cpSIdTags);

        break;
    case 4:     /* receive sIds */
        inputIdTags(stdin, &sIdTags);
#ifdef DEBUG
        outputIdTags(stderr, &sIdTags);
#endif
        shastraIdTagsOut(1 /* stdout */ , &sIdTags);

        break;
    }

}


void
shastraIdTagsXDRFree(pSIdTags)
    shastraIdTags  *pSIdTags;
{
    xdr_free(xdr_shastraIdTags, (char *) pSIdTags);
    memset(pSIdTags, 0, sizeof(shastraIdTags));
}

/*
 * Function --
 */
char           *
pSIdTag2Str(pSIdTag, fMask)
    shastraIdTag   *pSIdTag;
    int             fMask;
{
    /* if fMask, then convert Tag to Id and show that */
    char           *buf;
    int StrMaxLen = 16;

    buf = malloc(StrMaxLen);
    sprintf(buf, "%lu", *pSIdTag);
    return buf;
}


/*
 * Function --
```

```
 */
char          **
pSIdTags2StrTab(pSIdTags, fMask)
    shastraIdTags  *pSIdTags;
    int             fMask;
{
    int             i;
    char          **buf;

    buf = (char **) malloc(sizeof(char *) * (pSIdTags->shastraIdTags_len +
        1));
    for (i = 0; i < pSIdTags->shastraIdTags_len; i++) {
        buf[i] = pSIdTag2Str(&pSIdTags->shastraIdTags_val[i], fMask);
    }
    buf[pSIdTags->shastraIdTags_len] = NULL;
    return buf;
}
```

```
/*********************************************************************
    ***/
/*********************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*********************************************************************
    ***/
/*********************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/shastraIdTag.h>

bool_t
xdr_shastraIdTag(xdrs, objp)
    XDR *xdrs;
    shastraIdTag *objp;
{
    if (!xdr_u_long(xdrs, objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_shastraIdTags(xdrs, objp)
    XDR *xdrs;
    shastraIdTags *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->shastraIdTags_val, (u_int *)&objp-
        >shastraIdTags_len, ~0, sizeof(shastraIdTag), xdr_shastraIdTag)) {
        return (FALSE);
    }
    return (TRUE);
```

```
}

bool_t
xdr_shastraIdTags_P(xdrs, objp)
    XDR *xdrs;
    shastraIdTags_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(shastraIdTags),
        xdr_shastraIdTags)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>

#include <shastra/datacomm/videoImgH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

#define STANDALONEnn

int
videoImgOut(fd, pVImg)
    int             fd;
    videoImg       *pVImg;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_videoImg(&xdrs, pVImg)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
```

```
    if (!xdr_videoImg(mplexXDRSEnc(fd), pVImg)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
videoImgIn(fd, pVImg)
    int             fd;
    videoImg       *pVImg;
{
    XDR             xdrs;
    int             retVal = 0;

    videoImgXDRFree(pVImg);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_videoImg(&xdrs, pVImg)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_videoImg(mplexXDRSDec(fd), pVImg)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
videoImgMemOut(buf, size, pVImg)
    char           *buf;
    int             size;
    videoImg       *pVImg;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_videoImg(&xdrs, pVImg)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}
```

```
int
videoImgMemIn(buf, size, pVImg)
    char           *buf;
    int             size;
    videoImg       *pVImg;
{
    XDR             xdrs;
    int             retVal = 0;

    videoImgXDRFree(pVImg);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_videoImg(&xdrs, pVImg)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}
int
videoImgsOut(fd, pVImgs)
    int             fd;
    videoImgs      *pVImgs;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_videoImgs(&xdrs, pVImgs)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_videoImgs(mplexXDRSEnc(fd), pVImgs)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
videoImgsIn(fd, pVImgs)
    int             fd;
    videoImgs      *pVImgs;
{
    XDR             xdrs;
    int             retVal = 0;
```

```
    videoImgsXDRFree(pVImgs);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_videoImgs(&xdrs, pVImgs)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_videoImgs(mplexXDRSDec(fd), pVImgs)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}


int
videoImgsMemOut(buf, size, pVImgs)
    char           *buf;
    int             size;
    videoImgs      *pVImgs;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_videoImgs(&xdrs, pVImgs)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
videoImgsMemIn(buf, size, pVImgs)
    char           *buf;
    int             size;
    videoImgs      *pVImgs;
{
    XDR             xdrs;
    int             retVal = 0;

    videoImgsXDRFree(pVImgs);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_videoImg(&xdrs, pVImgs)) {
        retVal = -1;
    }
```

```
    xdr_destroy(&xdrs);
    return retVal;
}

int
videoClipOut(fd, pVClip)
    int             fd;
    videoClip      *pVClip;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_videoClip(&xdrs, pVClip)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_videoClip(mplexXDRSEnc(fd), pVClip)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
videoClipIn(fd, pVClip)
    int             fd;
    videoClip      *pVClip;
{
    XDR             xdrs;
    int             retVal = 0;

    videoClipXDRFree(pVClip);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_videoClip(&xdrs, pVClip)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
```

```
     */
    if (!xdr_videoClip(mplexXDRSDec(fd), pVClip)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}


int
videoClipMemOut(buf, size, pVClip)
    char           *buf;
    int             size;
    videoClip      *pVClip;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_videoClip(&xdrs, pVClip)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
videoClipMemIn(buf, size, pVClip)
    char           *buf;
    int             size;
    videoClip      *pVClip;
{
    XDR             xdrs;
    int             retVal = 0;

    videoClipXDRFree(pVClip);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_videoImg(&xdrs, pVClip)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}


void
freeVideoImg(pVImg)
    videoImg       *pVImg;
{
    if (pVImg == NULL) {
        return;
    }
    if (pVImg->data.data_val != NULL) {
```

```
        free(pVImg->data.data_val);
    }
    memset(pVImg, 0, sizeof(videoImg));
}


void
freeVideoImgs(pVImgs)
    videoImgs      *pVImgs;
{
    videoImgsXDRFree(pVImgs);
}

videoImg       *
copyVideoImg(pVImg, destpVImg)
    videoImg       *pVImg;
    videoImg       *destpVImg;
{
    videoImg       *newpVImg;
    int             i;

    if (pVImg == NULL) {
        return NULL;
    }
    if (destpVImg == NULL) {
        newpVImg = (videoImg *) malloc(sizeof(videoImg));
    } else {
        newpVImg = destpVImg;
    }

    memcpy(newpVImg, pVImg, sizeof(videoImg));
    newpVImg->data.data_val = (char *) malloc(newpVImg->data.data_len *
                    sizeof(newpVImg->data.data_val[0]));
    memcpy(newpVImg->data.data_val,pVImg->data.data_val,
          newpVImg->data.data_len * sizeof(newpVImg->data.data_val[0]));
    return newpVImg;
}

videoImgs      *
copyVideoImgs(pVImgs, destpVImgs)
    videoImgs      *pVImgs;
    videoImgs      *destpVImgs;
{
    int             i;
    videoImgs      *newpVImgs;

    if (pVImgs == NULL) {
        return NULL;
    }
    if (destpVImgs == NULL) {
        newpVImgs = (videoImgs *) malloc(sizeof(videoImgs));
    } else {
        newpVImgs = destpVImgs;
```

```
    }
    memcpy(newpVImgs, pVImgs, sizeof(videoImgs));
    newpVImgs->videoImgs_val = (videoImg *)
        malloc(newpVImgs->videoImgs_len * sizeof(videoImg));
    for (i = 0; i < newpVImgs->videoImgs_len; i++) {
        copyVideoImg(&pVImgs->videoImgs_val[i], &newpVImgs->videoImgs_val[i
            ]);
    }
    return newpVImgs;
}


void
inputVideoImg(fp, pVImg)
    FILE           *fp;
    videoImg       *pVImg;
{
    int             i, n;

    videoImgXDRFree(pVImg);
    fscanf(fp, "%ld", &pVImg->lIdTag);
    fscanf(fp, "%ld", &pVImg->lSIdTag);
    fscanf(fp, "%ld", &pVImg->lPerms);
    fscanf(fp, "%hd", &pVImg->imgFormat);
    fscanf(fp, "%hd", &pVImg->imgComp);
    fscanf(fp, "%hd", &pVImg->imgMode);
    fscanf(fp, "%hd", &pVImg->imgXSize);
    fscanf(fp, "%hd", &pVImg->imgYSize);
    fscanf(fp, "%hd", &pVImg->imgDepth);
    /*read colors*/
    fscanf(fp, "%d", &n);
    if(n > 0){
        pVImg->pColorMap = (viColorMap*)malloc(sizeof(viColorMap));
        pVImg->pColorMap->viColorMap_len = n;
        pVImg->pColorMap->viColorMap_val = (viColor*)malloc(
            n * sizeof(viColor));
        for (i = 0; i < n; i++) {
            fscanf(fp, "%hd%hd%hd",
                &pVImg->pColorMap->viColorMap_val[i][0],
                &pVImg->pColorMap->viColorMap_val[i][1],
                &pVImg->pColorMap->viColorMap_val[i][2]);
        }
    }
    fscanf(fp, "%d", &pVImg->data.data_len);
    pVImg->data.data_val = (char *) malloc(pVImg->data.data_len *
                        sizeof(pVImg->data.data_val[0]));
    for (i = 0; i < pVImg->data.data_len; i++) {
        fscanf(fp, "%d", &n);
        pVImg->data.data_val[i] = n;
    }
}

void
```

```
outputVideoImg(fp, pVImg)
    FILE           *fp;
    videoImg       *pVImg;
{
    int             i;

    fprintf(fp, "%ld\n", pVImg->lIdTag);
    fprintf(fp, "%ld\n", pVImg->lSIdTag);
    fprintf(fp, "%ld\n", pVImg->lPerms);
    fprintf(fp, "%hd\n", pVImg->imgFormat);
    fprintf(fp, "%hd\n", pVImg->imgComp);
    fprintf(fp, "%hd\n", pVImg->imgMode);
    fprintf(fp, "%hd\n", pVImg->imgXSize);
    fprintf(fp, "%hd\n", pVImg->imgYSize);
    fprintf(fp, "%hd\n", pVImg->imgDepth);
    if(pVImg->pColorMap != NULL){
        fprintf(fp, "%ld\n", pVImg->pColorMap->viColorMap_len);
        for (i = 0; i < pVImg->pColorMap->viColorMap_len; i++) {
            fprintf(fp, "%hd %hd %hd\n",
                pVImg->pColorMap->viColorMap_val[i][0],
                pVImg->pColorMap->viColorMap_val[i][1],
                pVImg->pColorMap->viColorMap_val[i][2]);
        }
    }
    fprintf(fp, "%ld\n", pVImg->data.data_len);
    for (i = 0; i < pVImg->data.data_len; i++) {
        if (!(i % 8)) {
            fprintf(fp, "\n");
        }
        fprintf(fp, "%d ", pVImg->data.data_val[i]);
    }
    fprintf(fp, "\n");
}


void
inputVideoImgs(fp, pVImgs)
    FILE           *fp;
    videoImgs      *pVImgs;
{
    int             i;

    videoImgsXDRFree(pVImgs);
    fscanf(fp, "%d", &pVImgs->videoImgs_len);
    pVImgs->videoImgs_val = (videoImg *)
        malloc(pVImgs->videoImgs_len * sizeof(videoImg));
    for (i = 0; i < pVImgs->videoImgs_len; i++) {
        inputVideoImg(fp, &pVImgs->videoImgs_val[i]);
    }
}

void
outputVideoImgs(fp, pVImgs)
```

```
    FILE           *fp;
    videoImgs      *pVImgs;
{
    int             i;

    fprintf(fp, "%d\n", pVImgs->videoImgs_len);
    for (i = 0; i < pVImgs->videoImgs_len; i++) {
        outputVideoImg(fp, &pVImgs->videoImgs_val[i]);
    }

}

void
videoImgXDRFree(pVImg)
    videoImg       *pVImg;
{
    xdr_free(xdr_videoImg, (char *) pVImg);
    memset(pVImg, 0, sizeof(videoImg));
}

void
videoImgsXDRFree(pVImgs)
    videoImgs      *pVImgs;
{
    xdr_free(xdr_videoImgs, (char *) pVImgs);
    memset(pVImgs, 0, sizeof(videoImgs));
}

void
videoClipXDRFree(pVClip)
    videoClip      *pVClip;
{
    xdr_free(xdr_videoClip, (char *) pVClip);
    memset(pVClip, 0, sizeof(videoClip));
}

#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
videoImgMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
    static videoImg vImg;
    static videoImgs vImgs;
    videoImgs      *cpVImgs;
    videoImg       *cpVImg;

    switch (argc) {
    case 1:     /* receive vImg */
        videoImgIn(0 /* stdin */ , &vImg);
        outputVideoImg(stdout, &vImg);
```

```
        cpVImg = copyVideoImg(&vImg, NULL);
        outputVideoImg(stdout, cpVImg);
        freeVideoImg(cpVImg);

        break;
    case 2:     /* receive vImg */
        inputVideoImg(stdin, &vImg);
#ifdef DEBUG
        outputVideoImg(stderr, &vImg);
#endif
        videoImgOut(1 /* stdout */ , &vImg);

        break;
    case 3:     /* receive vImgs */
        videoImgsIn(0 /* stdin */ , &vImgs);
        outputVideoImgs(stdout, &vImgs);
        cpVImgs = copyVideoImgs(&vImgs, NULL);
        outputVideoImgs(stdout, cpVImgs);
        freeVideoImgs(cpVImgs);

        break;
    case 4:     /* receive vImgs */
        inputVideoImgs(stdin, &vImgs);
#ifdef DEBUG
        outputVideoImgs(stderr, &vImgs);
#endif
        videoImgsOut(1 /* stdout */ , &vImgs);

        break;
    }

}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/videoImg.h>

bool_t
xdr_viColor(xdrs, objp)
    XDR *xdrs;
    viColor objp;
{
    if (!xdr_vector(xdrs, (char *)objp, 3, sizeof(u_short), xdr_u_short)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_viColorMap(xdrs, objp)
    XDR *xdrs;
    viColorMap *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->viColorMap_val, (u_int *)&objp->
        viColorMap_len, ~0, sizeof(viColor), xdr_viColor)) {
        return (FALSE);
    }
    return (TRUE);
```

```
}

bool_t
xdr_videoImg(xdrs, objp)
    XDR *xdrs;
    videoImg *objp;
{
    if (!xdr_u_long(xdrs, &objp->lIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lSIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lPerms)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgFormat)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgComp)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgMode)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgXSize)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgYSize)) {
        return (FALSE);
    }
    if (!xdr_u_short(xdrs, &objp->imgDepth)) {
        return (FALSE);
    }
    if (!xdr_pointer(xdrs, (char **)&objp->pColorMap, sizeof(viColorMap),
        xdr_viColorMap)) {
        return (FALSE);
    }
    if (!xdr_bytes(xdrs, (char **)&objp->data.data_val, (u_int *)&objp->
        data.data_len, ~0)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_videoImg_P(xdrs, objp)
    XDR *xdrs;
    videoImg_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(videoImg), xdr_videoImg))
        {
        return (FALSE);
```

```
    }
    return (TRUE);
}

bool_t
xdr_videoImgs(xdrs, objp)
    XDR *xdrs;
    videoImgs *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->videoImgs_val, (u_int *)&objp->
        videoImgs_len, ~0, sizeof(videoImg), xdr_videoImg)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_videoImgs_P(xdrs, objp)
    XDR *xdrs;
    videoImgs_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(videoImgs), xdr_videoImgs)
        ) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_videoClip(xdrs, objp)
    XDR *xdrs;
    videoClip *objp;
{
    if (!xdr_vector(xdrs, (char *)objp->sbName, 32, sizeof(char), xdr_char)
        ) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lSIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lPerms)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lType)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lMode)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lPointer)) {
```

```
        return (FALSE);
    }
    if (!xdr_pointer(xdrs, (char **)&objp->pVImgs, sizeof(videoImgs),
        xdr_videoImgs)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>

#include <shastra/datacomm/xsCntlDataH.h>
#include <shastra/network/mplex.h>
#include <shastra/network/rpc.h>

#define STANDALONEnn

int
xsCntlDataOut(fd, pXSData)
    int             fd;
    xsCntlData     *pXSData;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_xsCntlData(&xdrs, pXSData)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
```

```
    if (!xdr_xsCntlData(mplexXDRSEnc(fd), pXSData)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
xsCntlDataIn(fd, pXSData)
    int             fd;
    xsCntlData     *pXSData;
{
    XDR             xdrs;
    int             retVal = 0;

    xsCntlDataXDRFree(pXSData);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_xsCntlData(&xdrs, pXSData)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_xsCntlData(mplexXDRSDec(fd), pXSData)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
xsCntlDataMemOut(buf, size, pXSData)
    char           *buf;
    int             size;
    xsCntlData     *pXSData;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_xsCntlData(&xdrs, pXSData)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}
```

```
int
xsCntlDataMemIn(buf, size, pXSData)
    char           *buf;
    int             size;
    xsCntlData     *pXSData;
{
    XDR             xdrs;
    int             retVal = 0;

    xsCntlDataXDRFree(pXSData);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_xsCntlData(&xdrs, pXSData)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
xsCntlDatasOut(fd, pXSDatas)
    int             fd;
    xsCntlDatas    *pXSDatas;
{
    XDR             xdrs;
    int             retVal = 0;

#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdout /* fdopen(fd,"w") */ ;
        xdrstdio_create(&xdrs, fp, XDR_ENCODE);
        if (!xdr_xsCntlDatas(&xdrs, pXSDatas)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSEnc(fd), mplexOutStream(fd), XDR_ENCODE);
     */
    if (!xdr_xsCntlDatas(mplexXDRSEnc(fd), pXSDatas)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
xsCntlDatasIn(fd, pXSDatas)
    int             fd;
    xsCntlDatas    *pXSDatas;
{
    XDR             xdrs;
    int             retVal = 0;
```

```
    xsCntlDatasXDRFree(pXSDatas);
#ifdef STANDALONE
    {
        FILE           *fp;
        fp = stdin /* fdopen(fd,"r") */ ;
        xdrstdio_create(&xdrs, fp, XDR_DECODE);
        if (!xdr_xsCntlDatas(&xdrs, pXSDatas)) {
            retVal = -1;
        }
    }
#else               /* STANDALONE */
    /*
     * xdrstdio_create(mplexXDRSDec(fd), mplexInStream(fd), XDR_DECODE);
     */
    if (!xdr_xsCntlDatas(mplexXDRSDec(fd), pXSDatas)) {
        retVal = -1;
    }
#endif              /* STANDALONE */
    return retVal;
}

int
xsCntlDatasMemOut(buf, size, pXSDatas)
    char           *buf;
    int             size;
    xsCntlDatas    *pXSDatas;
{
    XDR             xdrs;
    int             retVal = 0;

    xdrmem_create(&xdrs, buf, size, XDR_ENCODE);
    if (!xdr_xsCntlDatas(&xdrs, pXSDatas)) {
        retVal = -1;
    }
    xdr_destroy(&xdrs);
    return retVal;
}

int
xsCntlDatasMemIn(buf, size, pXSDatas)
    char           *buf;
    int             size;
    xsCntlDatas    *pXSDatas;
{
    XDR             xdrs;
    int             retVal = 0;

    xsCntlDatasXDRFree(pXSDatas);
    xdrmem_create(&xdrs, buf, size, XDR_DECODE);
    if (!xdr_xsCntlDatas(&xdrs, pXSDatas)) {
        retVal = -1;
    }
```

```
    xdr_destroy(&xdrs);
    return retVal;
}

void
freeXSCntlData(pXSData)
    xsCntlData     *pXSData;
{
    if (pXSData == NULL) {
        return;
    }
    memset(pXSData, 0, sizeof(xsCntlData));
}


void
freeXSCntlDatas(pXSDatas)
    xsCntlDatas    *pXSDatas;
{
    int             i;

    if (pXSDatas == NULL) {
        return;
    }
    for (i = 0; i < pXSDatas->xsCntlDatas_len; i++) {
        freeXSCntlData(&pXSDatas->xsCntlDatas_val[i]);
    }
    free(pXSDatas->xsCntlDatas_val);
    memset(pXSDatas, 0, sizeof(xsCntlDatas));
}

xsCntlData     *
copyXSCntlData(pXSData, destpXSData)
    xsCntlData     *pXSData;
    xsCntlData     *destpXSData;
{
    xsCntlData     *newpXSData;
    int             i;

    if (pXSData == NULL) {
        return NULL;
    }
    if (destpXSData == NULL) {
        newpXSData = (xsCntlData *) malloc(sizeof(xsCntlData));
    } else {
        newpXSData = destpXSData;
    }

    memcpy(newpXSData, pXSData, sizeof(xsCntlData));
    return newpXSData;
}

xsCntlDatas    *
```

```
copyXSCntlDatas(pXSDatas, destpXSDatas)
    xsCntlDatas    *pXSDatas;
    xsCntlDatas    *destpXSDatas;
{
    int             i;
    xsCntlDatas    *newpXSDatas;
    char            buf[65536];

    if (pXSDatas == NULL) {
        return NULL;
    }
    if (destpXSDatas == NULL) {
        newpXSDatas = (xsCntlDatas *) malloc(sizeof(xsCntlDatas));
        memset(newpXSDatas, 0, sizeof(xsCntlDatas));
    } else {
        newpXSDatas = destpXSDatas;
    }
    xsCntlDatasMemOut(buf, 65536, pXSDatas);
    xsCntlDatasMemIn(buf, 65536, newpXSDatas);
    return newpXSDatas;
}


void
inputXSCntlData(fp, pXSData)
    FILE           *fp;
    xsCntlData     *pXSData;
{
    memset(pXSData, 0, sizeof(xsCntlData));
}

void
outputXSCntlData(fp, pXSData)
    FILE           *fp;
    xsCntlData     *pXSData;
{
    fprintf(stderr, "outputXSCntlData() not complete\n");
}


void
inputXSCntlDatas(fp, pXSDatas)
    FILE           *fp;
    xsCntlDatas    *pXSDatas;
{
    int             i;

    fscanf(fp, "%d", &pXSDatas->xsCntlDatas_len);
    pXSDatas->xsCntlDatas_val = (xsCntlData *)
        malloc(pXSDatas->xsCntlDatas_len * sizeof(xsCntlData));
    for (i = 0; i < pXSDatas->xsCntlDatas_len; i++) {
        inputXSCntlData(fp, &pXSDatas->xsCntlDatas_val[i]);
    }
```

```
}

void
outputXSCntlDatas(fp, pXSDatas)
    FILE           *fp;
    xsCntlDatas    *pXSDatas;
{
    int             i;

    fprintf(fp, "%d\n", pXSDatas->xsCntlDatas_len);
    for (i = 0; i < pXSDatas->xsCntlDatas_len; i++) {
        outputXSCntlData(fp, &pXSDatas->xsCntlDatas_val[i]);
    }

}

void
xsCntlDataXDRFree(pXSData)
    xsCntlData     *pXSData;
{
    xdr_free(xdr_xsCntlData, (char *) pXSData);
    memset(pXSData, 0, sizeof(xsCntlData));
}

void
xsCntlDatasXDRFree(pXSDatas)
    xsCntlDatas    *pXSDatas;
{
    xdr_free(xdr_xsCntlDatas, (char *) pXSDatas);
    memset(pXSDatas, 0, sizeof(xsCntlDatas));
}

#ifdef STANDALONE
main(argc, argv)
#else               /* STANDALONE */
xsCntlDataMain(argc, argv)
#endif              /* STANDALONE */
    int             argc;
    char          **argv;
{
    static xsCntlData xsCData;
    static xsCntlDatas xsCDatas;
    xsCntlDatas    *cpXSDatas;
    xsCntlData     *cpXSData;

    switch (argc) {
    case 1:     /* receive xsCntlData */
        xsCntlDataIn(0 /* stdin */ , &xsCData);
        outputXSCntlData(stdout, &xsCData);
        cpXSData = copyXSCntlData(&xsCData, NULL);
        outputXSCntlData(stdout, cpXSData);
        freeXSCntlData(cpXSData);
```

```
        break;
    case 2:     /* receive xsCntlData */
        inputXSCntlData(stdin, &xsCData);
#ifdef DEBUG
        outputXSCntlData(stderr, &xsCData);
#endif
        xsCntlDataOut(1 /* stdout */ , &xsCData);

        break;
    case 3:     /* receive xsCntlDatas */
        xsCntlDatasIn(0 /* stdin */ , &xsCDatas);
        outputXSCntlDatas(stdout, &xsCDatas);
        cpXSDatas = copyXSCntlDatas(&xsCDatas, NULL);
        outputXSCntlDatas(stdout, cpXSDatas);
        freeXSCntlDatas(cpXSDatas);

        break;
    case 4:     /* receive xsCntlDatas */
        inputXSCntlDatas(stdin, &xsCDatas);
#ifdef DEBUG
        outputXSCntlDatas(stderr, &xsCDatas);
#endif
        xsCntlDatasOut(1 /* stdout */ , &xsCDatas);

        break;
    }

}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */

#include <rpc/rpc.h>
#include <shastra/datacomm/xsCntlData.h>

bool_t
xdr_objId(xdrs, objp)
    XDR *xdrs;
    objId *objp;
{
    if (!xdr_u_long(xdrs, &objp->lSIdTag)) {
        return (FALSE);
    }
    if (!xdr_u_long(xdrs, &objp->lIdTag)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_objIds(xdrs, objp)
    XDR *xdrs;
    objIds *objp;
{
    if (!xdr_array(xdrs, (char **)&objp->objIds_val, (u_int *)&objp->
        objIds_len, ~0, sizeof(objId), xdr_objId)) {
```

```
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOpcode(xdrs, objp)
    XDR *xdrs;
    xsOpcode *objp;
{
    if (!xdr_enum(xdrs, (enum_t *)objp)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsMouseData(xdrs, objp)
    XDR *xdrs;
    xsMouseData *objp;
{
    if (!xdr_int(xdrs, &objp->event)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->x)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->y)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->value)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsPersData(xdrs, objp)
    XDR *xdrs;
    xsPersData *objp;
{
    if (!xdr_int(xdrs, &objp->fov)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->aspect)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->near)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->far)) {
        return (FALSE);
    }
```

```
    return (TRUE);
}

bool_t
xdr_xsOrthoData(xdrs, objp)
    XDR *xdrs;
    xsOrthoData *objp;
{
    if (!xdr_double(xdrs, &objp->left)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->right)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->bottom)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->top)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->near)) {
        return (FALSE);
    }
    if (!xdr_double(xdrs, &objp->far)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsViewData(xdrs, objp)
    XDR *xdrs;
    xsViewData *objp;
{
    if (!xdr_vector(xdrs, (char *)objp->eyePt, 3, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->refPt, 3, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->twist)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsPolarData(xdrs, objp)
    XDR *xdrs;
    xsPolarData *objp;
{
```

```
    if (!xdr_double(xdrs, &objp->distance)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->azim)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->inci)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->twist)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsF4vect(xdrs, objp)
    XDR *xdrs;
    xsF4vect objp;
{
    if (!xdr_vector(xdrs, (char *)objp, 4, sizeof(float), xdr_float)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsF4mat(xdrs, objp)
    XDR *xdrs;
    xsF4mat objp;
{
    if (!xdr_vector(xdrs, (char *)objp, 4, sizeof(xsF4vect), xdr_xsF4vect))
        {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsORGBData(xdrs, objp)
    XDR *xdrs;
    xsORGBData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->rgb, 3, sizeof(double), xdr_double)
        ) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
bool_t
xdr_xsOShadeData(xdrs, objp)
    XDR *xdrs;
    xsOShadeData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->shade)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsORGBMData(xdrs, objp)
    XDR *xdrs;
    xsORGBMData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->rgbModel)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOCMapData(xdrs, objp)
    XDR *xdrs;
    xsOCMapData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->colorMap)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsODispData(xdrs, objp)
    XDR *xdrs;
    xsODispData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_int(xdrs, &objp->dispMode)) {
        return (FALSE);
    }
```

```
    return (TRUE);
}

bool_t
xdr_xsOTransData(xdrs, objp)
    XDR *xdrs;
    xsOTransData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->trans, 3, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsORotateData(xdrs, objp)
    XDR *xdrs;
    xsORotateData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->rotate, 3, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOScaleData(xdrs, objp)
    XDR *xdrs;
    xsOScaleData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_vector(xdrs, (char *)objp->scale, 3, sizeof(double),
        xdr_double)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOAppMatData(xdrs, objp)
    XDR *xdrs;
    xsOAppMatData *objp;
{
```

```
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_xsF4mat(xdrs, objp->appMat)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOSetMatData(xdrs, objp)
    XDR *xdrs;
    xsOSetMatData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    if (!xdr_xsF4mat(xdrs, objp->setMat)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsOGroupData(xdrs, objp)
    XDR *xdrs;
    xsOGroupData *objp;
{
    if (!xdr_objIds(xdrs, &objp->objects)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsCntlData(xdrs, objp)
    XDR *xdrs;
    xsCntlData *objp;
{
    if (!xdr_xsOpcode(xdrs, &objp->opcode)) {
        return (FALSE);
    }
    switch (objp->opcode) {
    case xs_NoOp:
        break;
    case xs_Mouse:
        if (!xdr_xsMouseData(xdrs, &objp->xsCntlData_u.mouse)) {
            return (FALSE);
        }
        break;
    case xs_WinSelect:
        if (!xdr_objId(xdrs, &objp->xsCntlData_u.winIndex)) {
            return (FALSE);
```

```
    }
    break;
case xs_WinForeRGB:
    if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.foreRGB, 3, sizeof
        (double), xdr_double)) {
        return (FALSE);
    }
    break;
case xs_WinBackRGB:
    if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.backRGB, 3, sizeof
        (double), xdr_double)) {
        return (FALSE);
    }
    break;
case xs_WinShade:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.shade)) {
        return (FALSE);
    }
    break;
case xs_WinRGBModel:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.rgbModel)) {
        return (FALSE);
    }
    break;
case xs_WinColorMap:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.colorMap)) {
        return (FALSE);
    }
    break;
case xs_WinTexture:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.texture)) {
        return (FALSE);
    }
    break;
case xs_WinEnvMap:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.envMap)) {
        return (FALSE);
    }
    break;
case xs_WinDispMode:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.dispMode)) {
        return (FALSE);
    }
    break;
case xs_WinViewMode:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.viewMode)) {
        return (FALSE);
    }
    break;
case xs_WinPersMode:
    if (!xdr_int(xdrs, &objp->xsCntlData_u.persMode)) {
        return (FALSE);
    }
```

```
        break;
    case xs_WinSelMode:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.selMode)) {
            return (FALSE);
        }
        break;
    case xs_WinPers:
        if (!xdr_xsPersData(xdrs, &objp->xsCntlData_u.pers)) {
            return (FALSE);
        }
        break;
    case xs_WinOrtho:
        if (!xdr_xsOrthoData(xdrs, &objp->xsCntlData_u.ortho)) {
            return (FALSE);
        }
        break;
    case xs_WinView:
        if (!xdr_xsViewData(xdrs, &objp->xsCntlData_u.view)) {
            return (FALSE);
        }
        break;
    case xs_WinPolar:
        if (!xdr_xsPolarData(xdrs, &objp->xsCntlData_u.polar)) {
            return (FALSE);
        }
        break;
    case xs_WinViewEye:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.eyePt, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_WinViewRef:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.refPt, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_WinViewTwist:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.twist)) {
            return (FALSE);
        }
        break;
    case xs_WinTrans:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.trans, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_WinScale:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.scale, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
```

```
        }
        break;
    case xs_WinRotate:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.rotate, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_WinSetMat:
        if (!xdr_xsF4mat(xdrs, objp->xsCntlData_u.setMat)) {
            return (FALSE);
        }
        break;
    case xs_WinAppMat:
        if (!xdr_xsF4mat(xdrs, objp->xsCntlData_u.appMat)) {
            return (FALSE);
        }
        break;
    case xs_WinResetMat:
        break;
    case xs_ObjWireRGB:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.objRGB, 3, sizeof
            (double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_ObjShade:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objShade)) {
            return (FALSE);
        }
        break;
    case xs_ObjRGBModel:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objRgbModel)) {
            return (FALSE);
        }
        break;
    case xs_ObjColorMap:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objColorMap)) {
            return (FALSE);
        }
        break;
    case xs_ObjTexture:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objTexture)) {
            return (FALSE);
        }
        break;
    case xs_ObjEnvMap:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objEnvMap)) {
            return (FALSE);
        }
        break;
    case xs_ObjDispMode:
        if (!xdr_int(xdrs, &objp->xsCntlData_u.objDispMode)) {
```

```
            return (FALSE);
        }
        break;
    case xs_ObjTrans:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.objTrans, 3,
            sizeof(double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_ObjScale:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.objScale, 3,
            sizeof(double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_ObjRotate:
        if (!xdr_vector(xdrs, (char *)objp->xsCntlData_u.objRotate, 3,
            sizeof(double), xdr_double)) {
            return (FALSE);
        }
        break;
    case xs_ObjSetMat:
        if (!xdr_xsF4mat(xdrs, objp->xsCntlData_u.objSetMat)) {
            return (FALSE);
        }
        break;
    case xs_ObjAppMat:
        if (!xdr_xsF4mat(xdrs, objp->xsCntlData_u.objAppMat)) {
            return (FALSE);
        }
        break;
    case xs_ObjSetIndex:
        if (!xdr_objIds(xdrs, &objp->xsCntlData_u.objIndex)) {
            return (FALSE);
        }
        break;
    case xs_ObjDisplay:
        if (!xdr_objIds(xdrs, &objp->xsCntlData_u.objDisplay)) {
            return (FALSE);
        }
        break;
    case xs_ObjUndisplay:
        if (!xdr_objIds(xdrs, &objp->xsCntlData_u.objUndisplay)) {
            return (FALSE);
        }
        break;
    case xs_ObjDelete:
        if (!xdr_objIds(xdrs, &objp->xsCntlData_u.objDelete)) {
            return (FALSE);
        }
        break;
    case xs_ObjResetMat:
        break;
```

```
        case xs_ObjsWireRGB:
            if (!xdr_xsORGBData(xdrs, &objp->xsCntlData_u.objsRGB)) {
                return (FALSE);
            }
            break;
        case xs_ObjsShade:
            if (!xdr_xsOShadeData(xdrs, &objp->xsCntlData_u.objsShade)) {
                return (FALSE);
            }
            break;
        case xs_ObjsRGBModel:
            if (!xdr_xsORGBMData(xdrs, &objp->xsCntlData_u.objsRGBM)) {
                return (FALSE);
            }
            break;
        case xs_ObjsColorMap:
            if (!xdr_xsOCMapData(xdrs, &objp->xsCntlData_u.objsCMap)) {
                return (FALSE);
            }
            break;
        case xs_ObjsTexture:
            if (!xdr_xsOCMapData(xdrs, &objp->xsCntlData_u.objsTexture)) {
                return (FALSE);
            }
            break;
        case xs_ObjsEnvMap:
            if (!xdr_xsOCMapData(xdrs, &objp->xsCntlData_u.objsEnvMap)) {
                return (FALSE);
            }
            break;
        case xs_ObjsDispMode:
            if (!xdr_xsODispData(xdrs, &objp->xsCntlData_u.objsDispMode)) {
                return (FALSE);
            }
            break;
        case xs_ObjsTrans:
            if (!xdr_xsOTransData(xdrs, &objp->xsCntlData_u.objsTrans)) {
                return (FALSE);
            }
            break;
        case xs_ObjsRotate:
            if (!xdr_xsORotateData(xdrs, &objp->xsCntlData_u.objsRotate)) {
                return (FALSE);
            }
            break;
        case xs_ObjsScale:
            if (!xdr_xsOScaleData(xdrs, &objp->xsCntlData_u.objsScale)) {
                return (FALSE);
            }
            break;
        case xs_ObjsSetMat:
            if (!xdr_xsOSetMatData(xdrs, &objp->xsCntlData_u.objsSetMat)) {
                return (FALSE);
```

```
        }
        break;
    case xs_ObjsResetMat:
        if (!xdr_xsOGroupData(xdrs, &objp->xsCntlData_u.objsResetMat)) {
            return (FALSE);
        }
        break;
    case xs_ObjsAppMat:
        if (!xdr_xsOAppMatData(xdrs, &objp->xsCntlData_u.objsAppMat)) {
            return (FALSE);
        }
        break;
    case xs_ObjsSetIndex:
        if (!xdr_xsOGroupData(xdrs, &objp->xsCntlData_u.objsIndex)) {
            return (FALSE);
        }
        break;
    case xs_ObjsDisplay:
        if (!xdr_xsOGroupData(xdrs, &objp->xsCntlData_u.objsDisplay)) {
            return (FALSE);
        }
        break;
    case xs_ObjsUndisplay:
        if (!xdr_xsOGroupData(xdrs, &objp->xsCntlData_u.objsUndisplay)) {
            return (FALSE);
        }
        break;
    case xs_ObjsDelete:
        if (!xdr_xsOGroupData(xdrs, &objp->xsCntlData_u.objsDelete)) {
            return (FALSE);
        }
        break;
    }
    return (TRUE);
}

bool_t
xdr_xsCntlData_P(xdrs, objp)
    XDR *xdrs;
    xsCntlData_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(xsCntlData),
        xdr_xsCntlData)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsCntlDatas(xdrs, objp)
    XDR *xdrs;
    xsCntlDatas *objp;
{
```

```
    if (!xdr_array(xdrs, (char **)&objp->xsCntlDatas_val, (u_int *)&objp->
        xsCntlDatas_len, ~0, sizeof(xsCntlData), xdr_xsCntlData)) {
        return (FALSE);
    }
    return (TRUE);
}

bool_t
xdr_xsCntlDatas_P(xdrs, objp)
    XDR *xdrs;
    xsCntlDatas_P *objp;
{
    if (!xdr_pointer(xdrs, (char **)objp, sizeof(xsCntlDatas),
        xdr_xsCntlDatas)) {
        return (FALSE);
    }
    return (TRUE);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * clSvrCntl.c
 */
#include <stdio.h>
#include <string.h>

#include <shastra/shastra.h>

#include <shastra/utils/list.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/genui.h>

#include <shastra/shautils/clientHosts.h>
#include <shastra/shautils/kernelFronts.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>

#include <shastra/front/front.h>
#include <shastra/front/frontP.h>
#include <shastra/front/front_client.h>
#include <shastra/front/clSvrCntl.h>
#include <shastra/front/clSvrCntlP.h>
#include <shastra/front/shastraCntl.h>

static ShastraToolMode iClSvrModeMine;
static ShastraToolMode iClSvrMode;
```

```
static shastraId defServerSId = { NULL, NULL, TEST_SERVICE_NAME};
extern chooseOne     *pcoClSvr;
hostData       *pHostShaCurrClnt;
static shastraIdTag currClntSIdTag;


void
clSvrSetSelfModeOprn()
{
  iClSvrModeMine = shastraNameToMode(pFrontSId->nmApplicn);
}


char          **
getServerNameList(pSId)
    shastraId* pSId;
{
  char          **sbNames;

  if(pSId == NULL){
    if(iClSvrMode == 0){
      defServerSId.nmApplicn = pFrontSId->nmApplicn;
    }
    else{
      defServerSId.nmApplicn = shastraModeToName(iClSvrMode);
    }
    sbNames = clHosts2StrTab(&defServerSId, PSIDNMHOST | PSIDNMAPPL);
  }
  else{
    sbNames = clHosts2StrTab(pSId, PSIDNMHOST | PSIDNMAPPL);
  }
  return sbNames;
}

char          **
getServerNameListByService(iService)
     int iService;
{
  char          **sbNames;

  defServerSId.nmApplicn = shastraServiceToName(iService);
  sbNames = clHosts2StrTab(&defServerSId, PSIDNMHOST | PSIDNMAPPL);
  return sbNames;
}

void
setClSvrServerNamesOprn(pSId)
    shastraId *pSId;
{
  char          **sbNames, *sService;

  if(pcoClSvr == NULL){
    return;
```

```
  }
  sService = shastraModeToName(iClSvrMode);
  if(strcmp(pSId->nmApplicn,sService)){
    return; /*not current service type*/
  }
  sbNames = getServerNameList(pSId);
  chooseOneChangeList(pcoClSvr, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
}

/*
 * Function
 */
void
clSvrSetCurrHostOprn(pHost, fForce)
    hostData *pHost;
    int fForce;
{
  if(!fForce && (pHostShaCurrClnt != NULL)){
    return; /*only set if not already set*/
  }
  pHostShaCurrClnt = pHost;
  if(pHostShaCurrClnt != NULL){
    currClntSIdTag = pHostShaCurrClnt->lSIDTag;
#ifdef DEBUG
    fprintf(stderr,"currClntSIdTag = %ld, pHost = %ld\n",
        currClntSIdTag, pHost);
#endif /* DEBUG */
  }
  else{
    clSvrUnselectOprn();
  }
/*
  set and update user interface element flags.. mode etc
  */
}

/*
 * Function
 */
void
clSvrResetCurrHostOprn(pHost, fForce)
    hostData *pHost;
    int fForce;
{
  if(!fForce && (pHostShaCurrClnt != pHost)){
    return;         /*only set if not already set*/
  }
  else{
    clSvrUnselectOprn();
  }
```

```
}

hostData *
clSvrHostFromService(iService, iClSvr)
     int iService;
     int iClSvr;
{
  hostData       *pHost;

  defServerSId.nmApplicn = shastraServiceToName(iService);
  pHost = getClntHostByIndex(&defServerSId, iClSvr);

  return pHost;
}

hostData *
getClSvrHostFromIndex(iClSvr)
    int             iClSvr;
{
  hostData       *pHost;
  shastraId      *pSId = NULL;

  if(currClntSIdTag){
    pSId = mapSIdTag2SId(&currClntSIdTag);
  }
  if(pSId == NULL){
    pSId = &defServerSId;
    defServerSId.nmApplicn = shastraModeToName(iClSvrMode);
  }
  pHost = getClntHostByIndex(pSId, iClSvr);
#ifdef DEBUG
  fprintf(stderr,"getClSVrHostFromIndex()->smIdTag = %ld, pHost = %ld\n",
      pHost->lSIDTag, pHost);
#endif /* DEBUG */
  return pHost;
}

void
clSvrSetModeOprn(iMode)
    ShastraToolMode             iMode;
{
  iClSvrMode = iMode;
  /*update the shown set*/
  defServerSId.nmApplicn = shastraModeToName(iClSvrMode);

  setClSvrServerNamesOprn(&defServerSId);
}

/*
 * Function
 */
void
clSvrUnselectOprn()
```

```
{
  pHostShaCurrClnt = NULL;
  currClntSIdTag = 0;
}

/*
 * Function
 */
void
clSvrSelectOprn(i)
    int            i;
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  clSvrSetCurrHostOprn(pHost, True);
  if (clientSelectFunc != NULL) {
    (*clientSelectFunc) (pHostShaCurrClnt);
  }
}


/*
 * Function
 */
void
clSvrRenameOprn(i, name)
    int            i;
    char *name;
{
  /*change*/
}

/*
 * Function
 */
void
clSvrDisconnectOprn(i)
    int            i;
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  if(clntTerminateReq(NULL, pHost) == -1){
    clSvrUtilPopupMessage("clntTerminateReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
clSvrTerminateOprn(i)
    int            i;
```

```
{
  hostData *pHost;
  pHost = getClSvrHostFromIndex(i);
  if(clntTerminateReq(NULL, pHost) == -1){
    clSvrUtilPopupMessage("clntTerminateReq() Error!\n");
    return;
  }
  clSvrUtilPopupMessage("This operation is presently disabled!\n");
}

/*
 * Function
 */
void
clSvrCreateOprn(sbName)
    char *sbName;
{
  printf("create %s on %s\n", shastraModeToName(iClSvrMode), sbName);
  /*execute a starter script*/
}

/*
 * Function
 */
void
clSvrServerOprn(sbName, iPort)
    char *sbName;
    int iPort;
{
  shastraId sId;
  shaCmdData *pCmdData = NULL;

  if(!strcmp(pFrontSId->nmApplicn,sbName) &&
     (pFrontSId->iPort == iPort)){
    clSvrUtilPopupMessage("Warning: Connecting to self!\n");
  }
  memset(&sId, 0, sizeof(shastraId));
  sId.nmApplicn = shastraModeToName(iClSvrMode);
  sId.nmHost = sbName;
  sId.iPort = iPort;
/*CHECK*/
  sId.lSIDTag = mplexGetUniqueId();
  sId.lIPAddr = hostName2IPAddress(sbName);
  /*check if already connected*/
  if(getClntHostByIdTag(&sId, &sId.lSIDTag) != NULL){
    clSvrUtilPopupMessage("Warning: Already connected to host!\n");
  }
  printf("server connect to %s on %s\n", sId.nmApplicn, sbName);
  /* connect using non-shastra info */
  if(clientControlDataFunc){
    (*clientControlDataFunc)(shastraModeToService(iClSvrMode), &pCmdData);
    if(pCmdData == NULL){
      clSvrUtilPopupMessage("Invalid Control Data!\n");
```

```
      return;
    }
  }
  else{
    clSvrUtilPopupMessage("Can't Obtain Control Data!\n");
    return;
  }
  if(clntConnectReq(NULL, &sId, pCmdData) == -1){
    clSvrUtilPopupMessage("clntConnectReq() Error!\n");
    return;
  }
}


/*
 * Function
 */
void
clSvrConnectOprn(iWhich)
    int iWhich;
{
  shastraIdTag *pSIdTag;
  shastraId *pSId;
  shaCmdData *pCmdData = NULL;

  pSIdTag = krFrNdx2SIdTag(iWhich);
  pSId = mapSIdTag2SId(pSIdTag);
  if(pSId == NULL){
    clSvrUtilPopupMessage("Invalid System!\n");
    return;
  }
  if(*pSIdTag == pFrontSId->lSIDTag){
    clSvrUtilPopupMessage("Warning: Connecting to self!\n");
  }
  /*check if already connected*/
  if(getClntHostByIdTag(pSId, pSIdTag) != NULL){
    clSvrUtilPopupMessage("Warning: Already connected!\n");
  }
  if(clientControlDataFunc){
    (*clientControlDataFunc)(shastraNameToService(pSId->nmApplicn), &
        pCmdData);
    if(pCmdData == NULL){
      clSvrUtilPopupMessage("Invalid Control Data!\n");
      return;
    }
  }
  else{
    clSvrUtilPopupMessage("Can't Obtain Control Data!\n");
    return;
  }
  if(clntConnectReq(NULL, pSId, pCmdData) == -1){
    clSvrUtilPopupMessage("clntConnectReq() Error!\n");
    return;
```

```
  }
}


void
clSvrOperationsOprn(pMgrCD, fUp)
    mgrCntlData *pMgrCD;
    int fUp;
{
  if(pHostShaCurrClnt == NULL){
    clSvrUtilPopupMessage("Invalid Current Server!\n");
    return;
  }
  if (clientOperatorFunc != NULL) {
    (*clientOperatorFunc) (pHostShaCurrClnt);
  }
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * clSvrCntlUI.c
 */
#include <stdio.h>

#include <Xm/CascadeB.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleB.h>
#include <Xm/TextF.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Xm.h>
#include <X11/Shell.h>

#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/menu.h>
#include <shastra/uitools/toggles.h>
#include <shastra/uitools/buttons.h>
#include <shastra/uitools/genui.h>
#include <shastra/uitools/dialog.h>
#include <shastra/uitools/choose.h>
#include <shastra/uitools/text.h>
#include <shastra/uitools/controlPanel.h>

#include <shastra/front/frontState.h>
#include <shastra/front/frontP.h>
```

```
#include <shastra/front/clSvrCntl.h>
#include <shastra/front/clSvrCntlP.h>
#include <shastra/front/shastraCntl.h>

static void clSvrSysGenChooseOneSetup(Prot2(Widget, optChooseCntlData*));
static void clSvrSysGenChooseOneCB(Prot3(Widget, XtPointer, XtPointer));

Widget createHelpPD();

static Widget createClSvrControlPD();
static Widget createClSvrDebugPD();
static void clSvrSetModeCB();
static void clSvrShowTraceCB();
static void clSvrConnectCOCB();
static void clSvrOperationCB();
static void clSvrDismissCB();
static void clSvrDlgChooseCB();
static void chooseOneClSvrCB();
static void clSvrCmdCB();
static void clSvrConnectCB();


chooseOne *pcoClSvr;
static chooseOne *pcoClSvrSys;

static mgrCntlData *pClSvrDismissData;
static int fDebugTrace = 1;
static textCntlData clSvrMsgBufCntl = {"clSvrMsgBuffer", NULL, NULL};

static Widget
createClSvrMenuBar(wgParent, sName, argList)
     Widget          wgParent;
     char           *sName;
     XtVarArgsList   argList;
{
  Widget          wgMenuBar;
  Widget          wgToolPD, wgControlPD, wgDebugPD, wgHelpPD;
  Arg             args[8];
  int             n;
  static menuItem serversPD[] = {
    {"Shilp", (XtPointer) Shastra_OSHILP, False, clSvrSetModeCB},
    {"Ganith", (XtPointer) Shastra_OGANITH, False, clSvrSetModeCB},
    {"Vaidak", (XtPointer) Shastra_OVAIDAK, False, clSvrSetModeCB},
    {"Bhautik", (XtPointer) Shastra_OBHAUTIK, False, clSvrSetModeCB},
    {"Sculpt", (XtPointer) Shastra_OSCULPT, False, clSvrSetModeCB},
    {"Splinex", (XtPointer) Shastra_OSPLINEX, False, clSvrSetModeCB},
    {"Gati", (XtPointer) Shastra_OGATI, False, clSvrSetModeCB},
    {NULL}
  };
  static menuItem toolkitsPD[] = {
    {"Shilp", (XtPointer) Shastra_SHILP, False, clSvrSetModeCB},
    {"Ganith", (XtPointer) Shastra_GANITH, False, clSvrSetModeCB},
    {"Vaidak", (XtPointer) Shastra_VAIDAK, False, clSvrSetModeCB},
```

```
    {"Bhautik", (XtPointer) Shastra_BHAUTIK, False, clSvrSetModeCB},
    {"Sculpt", (XtPointer) Shastra_SCULPT, False, clSvrSetModeCB},
    {"Splinex", (XtPointer) Shastra_SPLINEX, False, clSvrSetModeCB},
    {"Gati", (XtPointer) Shastra_GATI, False, clSvrSetModeCB},
    {"Rasayan", (XtPointer) Shastra_RASAYAN, False, clSvrSetModeCB},
    {NULL}
  };
  static menuItem servicesPD[] = {
    {"Test", (XtPointer) Shastra_TEST, False, clSvrSetModeCB},
    {"Talk", (XtPointer) Shastra_TALK, False, clSvrSetModeCB},
    {"Draw", (XtPointer) Shastra_DRAW, False, clSvrSetModeCB},
    {"Poly", (XtPointer) Shastra_POLY, False, clSvrSetModeCB},
    {"Phone", (XtPointer) Shastra_PHONE, False, clSvrSetModeCB},
    {"Video", (XtPointer) Shastra_VIDEO, False, clSvrSetModeCB},
    {NULL}
  };
  static menuItem gamesPD[] = {
    {"Chess", (XtPointer) Shastra_CHESS, False, clSvrSetModeCB},
    {NULL}
  };
  static menuItem toolsPD[] = {
    {"Toolkits", NULL, False, NULL, NULL, NULL, toolkitsPD, MENU_RADIO_0},
    {"Services", NULL, False, NULL, NULL, NULL, servicesPD, MENU_RADIO_0},
    {"Games", NULL, False, NULL, NULL, NULL, gamesPD, MENU_RADIO_0},
    {"Servers", NULL, False, NULL, NULL, NULL, serversPD, MENU_RADIO_0},
    {NULL}
  };

  n = 0;
  if (argList) {
    XtSetArg(args[n], XtVaNestedList, argList);
    n++;
  }
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;

  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);
  wgControlPD = createClSvrControlPD(wgMenuBar);
  wgToolPD = pulldownMenuCreate(wgMenuBar, "Tools", MENU_CHECK,
                toolsPD, NULL);
  wgDebugPD = createClSvrDebugPD(wgMenuBar);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}

static void
createClSvrCntlAreaCB(wgParent, xpClient, xpCall)
     Widget          wgParent;
     XtPointer       xpClient, xpCall;
```

```
{
  Widget wgDbgText;
  Arg             args[16];
  int             n;

  n=0;
  XtSetArg(args[n], XmNrows, 5);n++;
  XtSetArg(args[n], XmNcolumns, 32);n++;
  XtSetArg(args[n], XmNeditable, False);n++;
  XtSetArg(args[n], XmNeditMode, XmMULTI_LINE_EDIT);n++;
  XtSetArg(args[n], XmNscrollingPolicy, XmAUTOMATIC); n++;
  XtSetArg(args[n], XmNvisualPolicy, XmCONSTANT); n++;
  XtSetArg(args[n], XmNscrollBarDisplayPolicy, XmAS_NEEDED); n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNscrollHorizontal, False); n++;
  XtSetArg(args[n], XmNscrollVertical, True); n++;
  XtSetArg(args[n], XmNwordWrap, True); n++;

  wgDbgText = createMessageBuffer(wgParent, "clSvrTextMsgs",
                  &clSvrMsgBufCntl, args,n);
  XtManageChild(wgDbgText);
}

static Widget
createClSvrDebugPD(wgMenuBar)
     Widget          wgMenuBar;
{
  Widget          wgDebugPD;
  static menuItem syncPD[] = {
    {"Foo", (XtPointer) NULL, False, NULL},
    {"Bar", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  static menuItem debugPD[] = {
    {"Sync.", NULL, False, NULL, NULL, NULL, syncPD, MENU_PUSH},
    {"Trace", (XtPointer) NULL, True, clSvrShowTraceCB, NULL,
         &xmToggleButtonWidgetClass},
    {NULL}
  };
    wgDebugPD = pulldownMenuCreate(wgMenuBar, "Debug", MENU_MIXED,
        debugPD, NULL);

    return wgDebugPD;
}

static void
clSvrShowTraceCB(wg, xpClient, cbs)
    Widget          wg;
    XtPointer       xpClient;
    XmToggleButtonCallbackStruct *cbs;
```

```
{
    fDebugTrace = cbs->set;
}

void
frontClSvrsCB(wgTgl, pMgrCD, xpFoo)
    Widget          wgTgl;
    mgrCntlData *pMgrCD;
    XtPointer xpFoo;
{
  Widget wgShell;
  panelCntlData       *pPanelCntl;
  int fToggles;
  static buttonItem panelBtns[] = {
    {"Create",(XtPointer)ClSvrCmd_CREATE, clSvrCmdCB},
    {"Server",(XtPointer)ClSvrCmd_SERVER, clSvrCmdCB},
    {"Connect",(XtPointer)ClSvrCmd_CONNECT, clSvrConnectCB},
    {NULL}
  };

  if (pMgrCD->wgCntl) {
    return;
  }
  pMgrCD->wgTgl = wgTgl;

  pPanelCntl = (panelCntlData *) malloc(sizeof(panelCntlData));
  memset(pPanelCntl, 0, sizeof(panelCntlData));

  pPanelCntl->sName = "ClSvr";
  pPanelCntl->fnMenuBar = createClSvrMenuBar;
  pPanelCntl->panelBtns = panelBtns;
  pPanelCntl->fnChooseCB = chooseOneClSvrCB;
  pPanelCntl->fCntlArea = True;

  fToggles = PANEL_SELECT | PANEL_UNSELECT |  PANEL_RENAME |
    PANEL_DISCONNECT | PANEL_TERMINATE ;
  pMgrCD->wgCntl = wgShell =
    createPanelControl(pMgrCD->wgParent, "serverControl",
              wgTgl, pPanelCntl,
              fToggles, PANEL_CHOOSEONE, NULL);

  createClSvrCntlAreaCB(pPanelCntl->wgCntlArea, NULL, NULL);

  pClSvrDismissData = pPanelCntl->pDismiss;
  pcoClSvr = pPanelCntl->pChooseOne;
}

static void
clSvrDismissCB(wg, xpClient, cbs)
    Widget          wg;
    XtPointer       xpClient;
    XmPushButtonCallbackStruct *cbs;
{
```

```
  defaultShellDismissCB(wg, (XtPointer)pClSvrDismissData, cbs);
}

static dialogCntlData dlgChooseClSvr;
static void
clSvrCmdCB(wg, xpClient, cbs)
    Widget wg;
    XtPointer xpClient;
    XmPushButtonCallbackStruct *cbs;
{
  dialogCntlData *pDialogCD = &dlgChooseClSvr;

  if(pDialogCD->wgDialog == NULL){
    Widget wgLabel, wgTextF;
    XmString str;

    pDialogCD->fFlags = DIALOG_OK | DIALOG_CANCEL | DIALOG_HELP;
    pDialogCD->fMode = XmDIALOG_APPLICATION_MODAL;
    pDialogCD->sMessage = "Choose a Host:";
    pDialogCD->sName = "Host";
    pDialogCD->fnCallback = clSvrDlgChooseCB;
    pDialogCD->fnNoMatchCallback = clSvrDlgChooseCB;
    pDialogCD->sbItems = NULL;

    createSelectionDialog(wg, "hostNameDialog", pDialogCD, NULL);
#ifdef WANT
    XtVaSetValues(pDialogCD->wgDialog,
          XmNchildPlacement, XmPLACE_BELOW_SELECTION,
          NULL);
    str = XmStringCreateSimple("Port Number");
    wgLabel =
      XtVaCreateManagedWidget("portLabel", xmLabelWidgetClass,
                  pDialogCD->wgDialog,
                  XmNalignment, XmALIGNMENT_BEGINNING,
                  XmNlabelString, str,
                  NULL);
    XmStringFree(str);
    wgTextF =
      XtVaCreateManagedWidget("portText", xmTextFieldWidgetClass,
                  pDialogCD->wgDialog,
                  XmNvalue, "0",
                  NULL);
#endif /*WANT*/
  }
  pDialogCD->xpClient = xpClient;
  defaultSelectionDialogPopup(pDialogCD, "Choose a Host:", "Host",
                  getHostNameList());
}

static void
clSvrDlgChooseCB(wg, xpClient, cbs)
    Widget wg;
    XtPointer xpClient;
```

```
    XmSelectionBoxCallbackStruct *cbs;
{
  dialogCntlData *pDialogCD = (dialogCntlData*)xpClient;
  ClSvrCmd iClSvrCmd = (ClSvrCmd)pDialogCD->xpClient;
  char *sName, *sPort = NULL;
  Widget wgList, wgTextF;
  static dialogCntlData dlgErrorMsgCD;
  dialogCntlData *pMsgDlgCD = &dlgErrorMsgCD;
  char nmBuf[256];
  int fKeepUp = 0, iPort = 0;

#ifdef WANT
  wgTextF = XmSelectionBoxGetChild(pDialogCD->wgDialog, XmDIALOG_WORK_AREA)
      ;
  if(wgTextF){
    sPort = XmTextFieldGetString(wgTextF);
    if(sPort){
      iPort = atoi(sPort);
      XtFree(sPort);
    }
    else{
      iPort = 0;
    }
  }
  iPort = (iPort < 0)? 0: iPort;
#endif /*WANT*/

  switch(cbs->reason){
  default:
    break;
  case XmCR_OK:
    XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &sName);
    switch(iClSvrCmd){
    case ClSvrCmd_CREATE:
      clSvrCreateOprn(sName);
      break;
    case ClSvrCmd_SERVER:
      clSvrServerOprn(sName, iPort);
      break;
    }
    XtFree(sName);
    break;
  case XmCR_NO_MATCH:
    XmStringGetLtoR(cbs->value, XmSTRING_DEFAULT_CHARSET, &sName);
    if(verifyHostNameOprn(sName)){
      wgList = XmSelectionBoxGetChild(wg, XmDIALOG_LIST);
      XmListAddItem(wgList, cbs->value, 0);
      switch(iClSvrCmd){
      case ClSvrCmd_CREATE:
    clSvrCreateOprn(sName);
    break;
      case ClSvrCmd_SERVER:
    clSvrServerOprn(sName, iPort);
```

```
        break;
          }
        }
        else{ /*tell invalid name*/
          if(pMsgDlgCD->wgDialog == NULL){
        pMsgDlgCD->fFlags = DIALOG_OK | DIALOG_HELP;
        pMsgDlgCD->fMode = XmDIALOG_APPLICATION_MODAL;
        pMsgDlgCD->sName = "Error";
        pMsgDlgCD->sMessage = "Bad Host!";

        createErrorDialog(wg, "errorDialog", pMsgDlgCD, NULL);
          }
          sprintf(nmBuf, "Unknown Host %s!", sName);
          defaultDialogPopupMessage(pMsgDlgCD, nmBuf);
          fKeepUp = 1;
        }
        XtFree(sName);
        break;
      }
      if(!fKeepUp){
        defaultDialogCancelCB(pDialogCD->wgDialog, (XtPointer)pDialogCD, cbs);
      }
    }

    static void
    disconnectClSvrsCB(wg, xpClient, cbs)
        Widget wg;
        XtPointer xpClient;
        XmAnyCallbackStruct *cbs;
    {
      panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;

      clSvrDisconnectOprn((int)pGenCD->xpCall);
    }

    static void
    terminateClSvrsCB(wg, xpClient, cbs)
        Widget wg;
        XtPointer xpClient;
        XmAnyCallbackStruct *cbs;
    {
      panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;

      clSvrTerminateOprn((int)pGenCD->xpCall);
    }

    static void
    renameClSvrsCB(wg, xpClient, cbs)
        Widget wg;
        XtPointer xpClient;
        XmAnyCallbackStruct *cbs;
    {
      panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;
```

```
  clSvrRenameOprn((int)pGenCD->xpCall, pGenCD->sbName);
}

static void
chooseOneClSvrCB(wg, xpClientData, xpCallData)
    Widget          wg;
    XtPointer       xpClientData, xpCallData;
{
  int            iWhich = (int ) xpCallData;
  panelCntlData       *pPanelCntl = (panelCntlData *) xpClientData;
  static panelAxnCntlData genCDConfirm;
  static panelAxnCntlData genCDRename;

  switch (pPanelCntl->iMode) {
  case PANEL_SELECT:
    clSvrSelectOprn(iWhich);
    break;
  case PANEL_UNSELECT:
/* old code   clSvrUnselectOprn(iWhich); */
    clSvrUnselectOprn();
    break;
  case PANEL_RENAME:
    genCDRename.fnCallback = renameClSvrsCB;
    genCDRename.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDRename;
    panelDefaultRenamePUCB(wg, xpClientData, xpCallData);
    break;
  case PANEL_DISCONNECT:
    genCDConfirm.fnCallback = disconnectClSvrsCB;
    genCDConfirm.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDConfirm;
    panelDefaultConfirmPUCB(wg, xpClientData, xpCallData);
    break;
  case PANEL_TERMINATE:
    genCDConfirm.fnCallback = terminateClSvrsCB;
    genCDConfirm.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDConfirm;
    panelDefaultConfirmPUCB(wg, xpClientData, xpCallData);
    break;
  default:
    break;
  }
}

static Widget
createClSvrControlPD(wgMenuBar)
    Widget          wgMenuBar;
{
  Widget          wgControlPD;
  static mgrCntlData cntlOperation;
  static menuItem controlPD[] = {
      {"Operations", (XtPointer) &cntlOperation, False, clSvrOperationCB},
```

```
        {"sep", (XtPointer) NULL, False, NULL, NULL, &xmSeparatorWidgetClass}
            ,
        {"Dismiss", (XtPointer) NULL, False, clSvrDismissCB},
        {NULL}
    };
    wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                    controlPD, NULL);

    return wgControlPD;
}


static void
clSvrOperationCB(wg, xpClient, cbs)
    Widget          wg;
    XtPointer       xpClient;
    XmToggleButtonCallbackStruct *cbs;
{
    mgrCntlData *mgrCntl = (mgrCntlData*)xpClient;

    clSvrOperationsOprn(mgrCntl, cbs->set);
}

static void
clSvrSetModeCB(wg, xpClient, cbs)
    Widget          wg;
    XtPointer       xpClient;
    XmToggleButtonCallbackStruct *cbs;
{
    static Widget wgCurrSet;

    if (cbs->set) {
      if(wgCurrSet && (wgCurrSet != wg)){
        XmToggleButtonSetState(wgCurrSet, False, True);
      }
      wgCurrSet = wg;
      clSvrSetModeOprn((ShastraToolMode)xpClient);
    }
}

static void
clSvrConnectCB(wg, xpClient, cbs)
    Widget wg;
    XtPointer xpClient;
    XmPushButtonCallbackStruct *cbs;
{
    static optChooseCntlData connectCD;
    connectCD.fnCallback = clSvrConnectCOCB;
    connectCD.xpClient = (XtPointer) NULL;
    clSvrSysGenChooseOneCB(wg, (XtPointer) & connectCD, NULL);
}

static void
```

```
clSvrConnectCOCB(wg, xpClient, xpCall)
    Widget          wg;
    XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  clSvrConnectOprn(iWhich);
}

/*
 * Function --
 */
void
clSvrShowInfo(s)
char *s;
{
  if(clSvrMsgBufCntl.wgText && fDebugTrace){
    wprintf(&clSvrMsgBufCntl,"%s", s);
  }
}

void
clSvrUtilPopupMessage(msg)
char *msg;
{
  static dialogCntlData infoDlgCD;

  clSvrShowInfo(msg);
  if(infoDlgCD.wgDialog == NULL){
    infoDlgCD.fFlags = DIALOG_OK;
    infoDlgCD.fBehave = DIALOG_AUTOLOWER;
    infoDlgCD.iDelay = 5000;
    infoDlgCD.sName = "Shastra Information";
    infoDlgCD.sMessage = "Yo, User Dude!\nThis is, like, cool!!";

    createInformationDialog(pFrontAppData->wgTop, "infoDialog",
                &infoDlgCD, NULL);
  }
  defaultDialogPopupMessage(&infoDlgCD, msg);
}

static void
clSvrSysGenChooseOneSetup(wg, pOptCD)
    Widget          wg;
    optChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static optChooseCntlData *pChooseOneCD;

  pChooseOneCD = pOptCD;

  if (pcoClSvrSys == NULL) {
    pcoClSvrSys = chooseOneCreate(asDef, coNoInitialHighlight,
```

```
                    wg, genCntlChooseCOCB,
                    (XtPointer) & pChooseOneCD, wg,
                    "Choose System", 200, NULL);
  }
}

static void
clSvrSysGenChooseOneCB(wg, xpClient, xpCall)
    Widget          wg;
    XtPointer       xpClient;
    XtPointer       xpCall;
{
  char          **sbNames;

  clSvrSysGenChooseOneSetup(wg, (optChooseCntlData *) xpClient);
  sbNames = getSystemNameList();
  chooseOneChangeList(pcoClSvrSys, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseOneMobExec(pcoClSvrSys, wg);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * collabCntl.c
 */
#include <stdio.h>
#include <string.h>

#include <shastra/utils/list.h>

#include <shastra/uitools/genui.h>
#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/text.h>
#include <shastra/uitools/dialog.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/clientHosts.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>

#include <shastra/front/collabCntl.h>
#include <shastra/front/collabCntlP.h>
#include <shastra/front/front.h>
#include <shastra/front/frontP.h>
#include <shastra/front/frontCollClient.h>
#include <shastra/front/front_client.h>
#include <shastra/front/frontState.h>
```

```
hostData *pHostShaCurrColl;
extern chooseOne *pcoCollab;
extern chooseOne *pcoCollabFronts;
static shastraId defSesmSId = { NULL, NULL, TESTSESM_SERVICE_NAME};

static shastraIdTag currCollSIdTag;
static CollabOptionState collOptionState;
static unsigned long collIxnMode;
static unsigned long collFloorMode;
static unsigned long collFormat;
static unsigned long collPermissions = SHASTRA_PERM_ACCESS |
SHASTRA_PERM_BROWSE | SHASTRA_PERM_MODIFY;
static int fFreeFloor = False;
static shastraIdTag sIdTagToken;

char          **
getCollabNameList(lSIdTag)
     shastraIdTag lSIdTag;
{
  char          **sbNames;
  shastraId      *pSId;

  if(lSIdTag == 0){
    lSIdTag = currCollSIdTag;
  }
  if(lSIdTag == 0){
    sbNames = clHosts2StrTab(&defSesmSId, PSIDNMHOST | PSIDNMAPPL);
  }
  else{
    pSId = getSIdByTagInSIds(&lSIdTag, &shastraSesmIds);
    sbNames = clHosts2StrTab(pSId, PSIDNMHOST | PSIDNMAPPL);
  }
  return sbNames;
}

char          **
getCollabFrontNameList(lSIdTag)
     shastraIdTag lSIdTag;
{
  char          **sbNames;
  shastraIdTags  *pSIdTags;

  if(lSIdTag == 0){
    lSIdTag = currCollSIdTag;
  }
  if(lSIdTag == 0){
    return NULL;
  }
  pSIdTags = getSesmFrontSIdTags(&lSIdTag);
  if(pSIdTags == NULL){
    /*shouldn't happen!*/
    return NULL;
  }
```

```
  sbNames = mapSIdTags2StrTab(pSIdTags,
                  PSIDNMHOST | PSIDNMAPPL | PSIDNMUSER);
  return sbNames;
}

void
setCollabNamesOprn(lSIdTag)
     shastraIdTag lSIdTag;
{
  char          **sbNames;
  int             iWhich;

  if(pcoCollab == NULL){
    return;
  }
  iWhich = getCollabIndex(lSIdTag);
  sbNames = getCollabNameList(lSIdTag);
  chooseOneChangeList(pcoCollab, sbNames, iWhich);
  if (sbNames) {
    strListDestroy(sbNames);
  }
}

void
setCollabFrontNamesOprn(lSIdTag)
     shastraIdTag lSIdTag;
{
  char          **sbNames;

  if((pcoCollabFronts == NULL) || (lSIdTag != currCollSIdTag)){
    return;
  }
  sbNames = getCollabFrontNameList(currCollSIdTag);
  chooseOneChangeList(pcoCollabFronts, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
}

void
setCollabFrontPermsOprn(lSIdTag)
     shastraIdTag lSIdTag;
{
  unsigned long lPerms;

  if((pcoCollab == NULL) || (lSIdTag != currCollSIdTag)){
    return;
  }
  lPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  collabSetPermToggles(lPerms);
}

void
```

```
setCollabFrontFloorOprn(smSIdTag, lSIdTag)
     shastraIdTag smSIdTag, lSIdTag;
{
  unsigned long lPerms;
  char *sName;
  int fHave, fFree;

  if((pcoCollab == NULL) || (smSIdTag != currCollSIdTag)){
    return;
  }
  sIdTagToken = lSIdTag;
  sName = mapSIdTag2Str(&lSIdTag, PSIDNMHOST | PSIDNMUSER);
  fHave = (lSIdTag == pFrontSId->lSIDTag);
  fFree = False;
  collabSetFloorInfo(sName, fHave, fFree);
  free(sName);
}

int
getCollabIndex(lSIdTag)
     shastraIdTag   lSIdTag;
{
  int             iSession;
  shastraId      *pSId;

  lSIdTag = (lSIdTag == 0) ? currCollSIdTag : lSIdTag;
  pSId = getSIdByTagInSIds(&lSIdTag, &shastraSesmIds);
  if(pSId != NULL){
    iSession = clHostsGetSIdTagIndex(pSId, &lSIdTag);
  }
  else{
    iSession = -1;
  }
  return iSession;
}

shastraIdTag
getCollabSIdTagFromIndex(iSession)
     int             iSession;
{
  shastraId      *pSId = NULL;
  shastraIdTags  *pSIdTags;

  if(currCollSIdTag){
    pSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  }
  if(pSId == NULL){
    pSId = &defSesmSId; /*WONT WORK FOR OTHER SESSION TYPES*/
  }
  pSIdTags = getClntHostSIdTags(pSId);
  if((iSession < 0) || (iSession >= pSIdTags->shastraIdTags_len)){
    return 0;
  }
```

```
  else{
    return pSIdTags->shastraIdTags_val[iSession];
  }
}

hostData *
getCollabHostFromIndex(iSession)
     int             iSession;
{
  hostData       *pHost;
  shastraId      *pSId = NULL;

  if(currCollSIdTag){
    pSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  }
  if(pSId == NULL){
    pSId = &defSesmSId; /*WONT WORK FOR OTHER SESSION TYPES*/
  }
  pHost = getClntHostByIndex(pSId, iSession);
#ifdef DEBUG
  fprintf(stderr,"getCollabHostFromIndex()->smIdTag = %ld, pHost = %ld\n",
      pHost->lSIDTag, pHost);
#endif /* DEBUG */
  return pHost;
}

/*
 * Function
 */
void
collabSetCurrHostOprn(pHost, fForce)
     hostData *pHost;
     int fForce;
{
  if(!fForce && (pHostShaCurrColl != NULL)){
    return; /*only set if not already set*/
  }
  pHostShaCurrColl = pHost;
  if(pHostShaCurrColl != NULL){
    currCollSIdTag = pHostShaCurrColl->lSIDTag;
#ifdef DEBUG
    fprintf(stderr,"currCollSIdTag = %ld, pHost = %ld\n",
        currCollSIdTag, pHost);
#endif /* DEBUG */
  }
  else{
    collabUnselectOprn(0);
  }
  if(currCollSIdTag){
    setCollabFrontNamesOprn(currCollSIdTag);
  }
/*
  set and update user interface element flags.. sIdTagToken, perms, etc etc
```

```
  */
}

/*
 * Function
 */
void
collabResetCurrHostOprn(pHost, fForce)
     hostData *pHost;
     int fForce;
{
  if(!fForce && (pHostShaCurrColl != pHost)){
    return; /*only set if not already set*/
  }
  else{
    collabUnselectOprn(0);
  }
}

/*
 * Function
 */
void
collabSelectOprn(i)
     int            i;
{
  hostData *pHost;
  pHost = getCollabHostFromIndex(i);
  collabSetCurrHostOprn(pHost, True);
  if (collabSelectFunc != NULL) {
    (*collabSelectFunc) (pHostShaCurrColl);
  }
}

/*
 * Function
 */
void
collabUnselectOprn(i)
     int            i;
{
  pHostShaCurrColl = NULL;
  currCollSIdTag = 0;
}

/*
 * Function
 */
void
collabRenameOprn(i, iMode)
     int            i;
     int            iMode;
{
```

```
/*change*/
}

/*
 * Function
 */
void
collabLeaveOprn(i)
     int            i;
{
  hostData *pHost;

  pHost = getCollabHostFromIndex(i);
  if (pHost == NULL) {
    collabUtilPopupMessage("Invalid Session!\n");
    return;
  }
  if(collLeaveReq(pHost) == -1){
    collabUtilPopupMessage("collLeaveReq() Error!\n");
    return;
  }
  if(pHost->lSIDTag == currCollSIdTag){
    collabUnselectOprn(0);
  }
}

/*
 * Function
 */
void
collabTerminateOprn(i)
     int            i;
{
  hostData *pHost;
  unsigned long   myPerms;

  pHost = getCollabHostFromIndex(i);
  if (pHost == NULL) {
    collabUtilPopupMessage("Invalid Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Terminate!\n");
    return;
  }
  if(collTerminateReq(pHost) == -1){
    collabUtilPopupMessage("collTerminateReq() Error!\n");
    return;
  }
  if(pHost->lSIDTag == currCollSIdTag){
    collabUnselectOprn(0);
  }
```

```
}

/*
 * Function
 */
void
collabInitiateOprn(aiWhich)
     int *aiWhich;
{
  shastraIdTags *pSIdTags;
  shastraIdTag lSIdTag;
  shastraIdTag *pSIdTag;
  shaCommCntlData *pCommCD;
  int             i;

  pSIdTag = &pFrontSId->lSIDTag;
  pSIdTags = (shastraIdTags*)malloc(sizeof(shastraIdTags));
  memset(pSIdTags, 0, sizeof(shastraIdTags));
  krFrNdxs2SIdTags(aiWhich, pSIdTags);

  /* here make sure I myself am the first tag on the list */
  i = getSIdTagIndexInSIdTags(pSIdTag, pSIdTags);
  if (i == -1) {        /* not in this */
    addSIdTag2SIdTags(pSIdTag, pSIdTags);
    i = getSIdTagIndexInSIdTags(pSIdTag, pSIdTags);
  }
  if (i > 0) {      /* exchange first with this */
    pSIdTags->shastraIdTags_val[i] = pSIdTags->shastraIdTags_val[0];
    pSIdTags->shastraIdTags_val[0] = pFrontSId->lSIDTag;
  }
  lSIdTag = mplexGetUniqueId();
  if(collOptionState.iForceJoinOpt == CollabOpt_FORCEJOIN){
    if(collAutoInitiateReq(pHostKernel, pSIdTags, collPermissions,
               lSIdTag) == -1){
      collabUtilPopupMessage("collAutoInitiateReq() Error!\n");
      return;
    }
  }
  else{
    if(collInitiateReq(pHostKernel, pSIdTags, collPermissions,
               lSIdTag) == -1){
      collabUtilPopupMessage("collInitiateReq() Error!\n");
      return;
    }
  }
  if(pSIdTags->shastraIdTags_len == 1){
    return;
  }
  if((pCommCD = getCollabCommData(lSIdTag, pFrontSId->lSIDTag,
                  ShaComm_INITIATE)) == NULL){
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
```

```
    pCommCD->remSIdTag = lSIdTag;
    pCommCD->smSIdTag = lSIdTag;
    pCommCD->pSIdTags = pSIdTags;
    pCommCD->fShowList = True;
    pCommCD->iCommMode = ShaComm_INVITE;
    setupCollabInviteCommDialog(pCommCD);
    setCollabCommData(lSIdTag, pFrontSId->lSIDTag,
              ShaComm_INITIATE, pCommCD);
  }
  defaultDialogPopup(pCommCD->pDialogCD);
}

void
collabDeleteInvitePanelOprn(smSIdTag)
     shastraIdTag smSIdTag;
{
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                  ShaComm_INVITE);
  if(pCommCD != NULL){
    freeCollabCommData(smSIdTag, pFrontSId->lSIDTag, ShaComm_INVITE);
    free(pCommCD);
  }
}

/*
 * Function
 */
void
collabSetLeaderOprn(sIdTag, smSIdTag, lIdTag)
     shastraIdTag sIdTag;
     shastraIdTag smSIdTag;
     unsigned long lIdTag;
{
  shaCommCntlData *pCommCD;

  if(sIdTag != pFrontSId->lSIDTag){
    fprintf(stderr,"collabSetLeaderOprn()-> not for me!!\n");
  }
  if(lIdTag == 0){
    return; /*nothing needs to happen*/
  }
  pCommCD = getCollabCommData(lIdTag, sIdTag, ShaComm_INITIATE);
  if(pCommCD != NULL){
    freeCollabCommData(lIdTag, sIdTag, ShaComm_INITIATE);
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->remSIdTag = smSIdTag;
    setCollabCommData(smSIdTag, sIdTag, ShaComm_INVITE, pCommCD);
    shastraCommAppendText(pCommCD, "\nThis Front is the Group Leader!\n");
  }
  else{
    collabUtilPopupMessage("This Front is the Group Leader!\n");
```

```
  }
  setCollabFrontPermsOprn(smSIdTag);
}

/*
 * Function
 */
void
collabInviteOprn(aiWhich)
     int *aiWhich;
{
  int i, n;
  shastraIdTags *pSIdTags;
  shastraId *pSesmSId;
  shaCommCntlData *pCommCD;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Invite!\n");
    return;
  }
  pSIdTags = (shastraIdTags*)malloc(sizeof(shastraIdTags));
  memset(pSIdTags, 0, sizeof(shastraIdTags));
  krFrNdxs2SIdTags(aiWhich, pSIdTags);

  if(pSIdTags->shastraIdTags_len == 0){
    free(pSIdTags);
    return;
  }

  if(collOptionState.iForceJoinOpt == CollabOpt_FORCEJOIN){ }
  else{ }

  for(i=0, n = 0;i <pSIdTags->shastraIdTags_len;i++){
    if(!frontIsInCollSession(pSIdTags->shastraIdTags_val[i],
                currCollSIdTag)){
      n++;
      if(collInviteJoinReq(pHostKernel, &currCollSIdTag,
               &pSIdTags->shastraIdTags_val[i],
               &pFrontSId->lSIDTag,
               (shastraIdTag*)&collPermissions) == -1){
    collabUtilPopupMessage("collInviteJoinReq() Error!\n");
    return;
      }
    }
    else{
      collabUtilPopupMessage("System Already In Session!\n");
```

```
      }
    }
    if(n == 0){
      return;
    }
    if((pCommCD = getCollabCommData(currCollSIdTag, pFrontSId->lSIDTag,
                    ShaComm_INVITE)) == NULL){
      pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
      memset(pCommCD, 0, sizeof(shaCommCntlData));
      pCommCD->locSIdTag = pFrontSId->lSIDTag;
      pCommCD->remSIdTag = currCollSIdTag;
      pCommCD->smSIdTag = currCollSIdTag;
      pCommCD->pSIdTags = pSIdTags;
      pCommCD->fShowList = True;
      pCommCD->iCommMode = ShaComm_INVITE;
      setupCollabInviteCommDialog(pCommCD);
      setCollabCommData(currCollSIdTag, pFrontSId->lSIDTag,
                ShaComm_INVITE, pCommCD);
    }
    defaultDialogPopup(pCommCD->pDialogCD);
}

/*
 * Function
 */
void
collabSetInviteStatusOprn(smSIdTag, toSIdTag, lStatus)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
     unsigned long lStatus;
{
  shastraId *pSesmSId;
  shastraId *pRemSId;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Inviter Session!\n");
    return;
  }
  pRemSId = krFrSIdTag2SId(toSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Inviter Session Leader!\n");
    return;
  }
  if(collInviteStatusReq(pHostKernel, &smSIdTag, &toSIdTag,
             &pFrontSId->lSIDTag,
             lStatus) == -1){
    collabUtilPopupMessage("collInviteStatusReq() Error!\n");
    return;
  }
}

/*
```

```
 * Function
 */
void
collabJoinOprn(smSIdTag, permTag)
     shastraIdTag    smSIdTag;
     shastraIdTag    permTag;
{
  shastraId      *pSId;
  shastraIdTag   *pSIdTag;
  shaCmdData *pCmdData = NULL;

  pSIdTag =  & pFrontSId->lSIDTag;
  pSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSId == NULL){
    collabUtilPopupMessage("Invalid Session!\n");
    return;
  }
  if (frontIsInCollSession(*pSIdTag, smSIdTag)) {
    collabUtilPopupMessage("Already In Session!\n");
    return;
  }
  /* disallow multiple connexns to same sesMgr */
  if(collabControlDataFunc){
    (*collabControlDataFunc)(shastraNameToService(pSId->nmApplicn), &
        pCmdData);
    if(pCmdData == NULL){
      collabUtilPopupMessage("Invalid Control Data!\n");
      return;
    }
  }
  else{
    collabUtilPopupMessage("Can't Obtain Control Data!\n");
    return;
  }
  if(collJoinReq((hostData*)NULL, pSId, &permTag, pCmdData) == -1){
    collabUtilPopupMessage("collJoinReq() Error!\n");
    return;
  }
}

void
collabInviteAcceptOprn(smSIdTag, ldrSIdTag)
     shastraIdTag smSIdTag;
     shastraIdTag ldrSIdTag;
{
  unsigned long lStatus = 1;
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, ldrSIdTag, ShaComm_INVRESP);
  if(pCommCD != NULL){
    collabSetInviteStatusOprn(smSIdTag, ldrSIdTag, lStatus);
    collabJoinOprn(smSIdTag, pCommCD->lPerms);
    freeCollabCommData(smSIdTag, ldrSIdTag, ShaComm_INVRESP);
```

```
    free(pCommCD);
  }
  else{
    collabJoinOprn(smSIdTag, 0xff);
  }
}

void
collabInviteDeclineOprn(smSIdTag, ldrSIdTag)
     shastraIdTag smSIdTag;
     shastraIdTag ldrSIdTag;
{
  unsigned long lStatus = 0;
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, ldrSIdTag, ShaComm_INVRESP);
  if(pCommCD != NULL){
    collabSetInviteStatusOprn(smSIdTag, ldrSIdTag, lStatus);
    freeCollabCommData(smSIdTag, ldrSIdTag, ShaComm_INVRESP);
    free(pCommCD);
  }
}

/*
 * Function
 */
void
collabInvitePromptOprn(smSIdTag, leaderSIdTag, frontPerms)
     shastraIdTag smSIdTag, leaderSIdTag;
     unsigned long frontPerms;
{
  shastraIdTags *pSIdTags;
  shastraId *pSesmSId, *pRemSId;
  shaCommCntlData *pCommCD;
  unsigned long lRespStatus;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Invite Session!\n");
    return;
  }
  pRemSId = krFrSIdTag2SId(leaderSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Session Leader!\n");
    return;
  }

  if(frontIsInCollSession(pFrontSId->lSIDTag, smSIdTag)){
    collabUtilPopupMessage("System Already In Session!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&smSIdTag);
```

```
  switch(collOptionState.iInviteOpt){
  case CollabOpt_ACCEPT:
    collabUtilPopupMessage("Automatically Accepted Session Invitation!\n");
    lRespStatus = 1;
    if(collInviteStatusReq(pHostKernel, &smSIdTag, &leaderSIdTag,
              &pFrontSId->lSIDTag, lRespStatus) == -1){
      collabUtilPopupMessage("collInviteStatusReq() Error!\n");
      return;
    }
    collabJoinOprn(smSIdTag, frontPerms);
    return;
    break;
  case CollabOpt_DECLINE:
    collabUtilPopupMessage("Automatically Declined Session Invitation!\n");
    lRespStatus = 0;
    if(collInviteStatusReq(pHostKernel, &smSIdTag, &leaderSIdTag,
              &pFrontSId->lSIDTag, lRespStatus) == -1){
      collabUtilPopupMessage("collInviteStatusReq() Error!\n");
      return;
    }
    return;
    break;
  default:
    break;
  }

  if((pCommCD = getCollabCommData(smSIdTag, leaderSIdTag, ShaComm_INVRESP))
     == NULL){
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
    pCommCD->remSIdTag = leaderSIdTag;
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->lPerms = frontPerms;
    pCommCD->pSIdTags = pSIdTags;
    pCommCD->fShowList = True;
    pCommCD->iCommMode = ShaComm_INVRESP;
    setupCollabInvRespCommDialog(pCommCD);
    setCollabCommData(smSIdTag, leaderSIdTag, ShaComm_INVRESP, pCommCD);
  }
  /*got another invite for same conference.. ignore??*/
  defaultDialogPopup(pCommCD->pDialogCD);
}

/*
 * Function
 */
void
collabShowInviteStatusOprn(smSIdTag, toSIdTag, sIdTag, lStatus)
     shastraIdTag smSIdTag,toSIdTag,sIdTag;
     unsigned long lStatus;
{
  char msgBuf[256];
```

```
  char *sName;
  shaCommCntlData *pCommCD;

  sName = mapSIdTag2Str(&sIdTag, PSIDNMHOST | PSIDNMAPPL | PSIDNMUSER);

  if(lStatus){
    sprintf(msgBuf,"(%s)\n has accepted invitation\n", sName);
  }
  else{
    sprintf(msgBuf,"(%s)\n has declined invitation\n", sName);
  }
  free(sName);
  if((pCommCD = getCollabCommData(smSIdTag, toSIdTag, ShaComm_INVITE))
     != NULL){
    shastraCommAppendText(pCommCD, msgBuf);
  }
  else{
    collabUtilPopupMessage(msgBuf);
  }
}

void
collabSendInviteMessageOprn(smSIdTag, pToSIdTags, msg)
     shastraIdTag smSIdTag;
     shastraIdTags *pToSIdTags;
     char *msg;
{
  shastraIdTag toSIdTag;
  shastraId *pSesmSId;
  int i;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    /*unique-id not yet clobbered*/
    collabUtilPopupMessage("Session not Started.. Please Retry!\n");
    return;
  }
  if((pToSIdTags == NULL) || (pToSIdTags->shastraIdTags_len == 0)){
    collabUtilPopupMessage("Null Recipients for Invite Message!\n");
    return;
  }
  for(i=0; i< pToSIdTags->shastraIdTags_len;i++){
    toSIdTag = pToSIdTags->shastraIdTags_val[i];
    if(toSIdTag != pFrontSId->lSIDTag){
      if(collInviteMsgReq(pHostKernel, &smSIdTag, &toSIdTag,
               &pFrontSId->lSIDTag, msg) == -1){
    collabUtilPopupMessage("collInviteMsgReq() Error!\n");
    return;
      }
    }
  }
}
```

```
void
collabSendUniInviteMessageOprn(smSIdTag, toSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
     char *msg;
{
  shastraId *pSesmSId;
  int i;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    /*unique-id not yet clobbered*/
    collabUtilPopupMessage("Session not Started.. Please Retry!\n");
    return;
  }
  if(toSIdTag != pFrontSId->lSIDTag){
    if(collInviteMsgReq(pHostKernel, &smSIdTag, &toSIdTag,
            &pFrontSId->lSIDTag, msg) == -1){
      collabUtilPopupMessage("collInviteMsgReq() Error!\n");
      return;
    }
  }
}

void
collabRecvdInviteMessageOprn(smSIdTag, fromSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag fromSIdTag;
     char *msg;
{
  shaCommCntlData *pCommCD;

  if((pCommCD = getCollabCommData(smSIdTag, fromSIdTag, ShaComm_INVRESP))
     != NULL){
    /*should've been prompted, so if no panel, commited*/
    shastraCommDisplayText(pCommCD, msg);
  }
}

void
collabSendInvRespMessageOprn(smSIdTag, toSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
     char *msg;
{
  shastraId *pSesmSId;
  shastraId *pRemSId;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Inviter Session!\n");
    return;
  }
```

```
  pRemSId = krFrSIdTag2SId(toSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Inviter Session Leader!\n");
    return;
  }
  if(collInvRespMsgReq(pHostKernel, &smSIdTag, &toSIdTag,
              &pFrontSId->lSIDTag, msg) == -1){
    collabUtilPopupMessage("collInvRespMsgReq() Error!\n");
    return;
  }
}

void
collabRecvdInvRespMessageOprn(smSIdTag, fromSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag fromSIdTag;
     char *msg;
{
  shaCommCntlData *pCommCD;

  /*many such messages may come from the invitees FIX*/
  if((pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                 ShaComm_INVITE)) != NULL){
    /*also check if i'm in the SIdTag list, else ignore*/
    shastraCommAppendText(pCommCD, msg);
  }
#ifdef WANTSEPARATEPANELS
  if((pCommCD = getCollabCommData(smSIdTag, fromSIdTag, ShaComm_UNIINVRESP)
      )
     == NULL){
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
    pCommCD->remSIdTag = fromSIdTag;
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->iCommMode = ShaComm_UNIINVRESP;
    setupCollabInviteCommDialog(pCommCD);
    setCollabCommData(smSIdTag, fromSIdTag, ShaComm_UNIINVRESP, pCommCD);
  }
  shastraCommDisplayText(pCommCD, msg);
#endif /* WANTSEPARATEPANELS */
}

/*
 * Function
 */
void
collabAskJoinOprn(i)
     int i;
{
  shastraIdTag smSIdTag;
  shaCommCntlData *pCommCD;
  shastraId *pSesmSId;
```

```
  if ((i < 0) || (i >= shastraSesmIds.shastraIds_len)){
    return;
  }
  smSIdTag = shastraSesmIds.shastraIds_val[i]->lSIDTag;
  if(frontIsInCollSession(pFrontSId->lSIDTag, smSIdTag)){
    collabUtilPopupMessage("System Already In Session!\n");
    return;
  }
  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Ask-Join Session!\n");
    return;
  }
  if(collAskJoinReq(pHostKernel, &smSIdTag,
            &pFrontSId->lSIDTag) == -1){
    collabUtilPopupMessage("collAskJoinReq() Error!\n");
    return;
  }

  if((pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                  ShaComm_ASKJOIN)) == NULL){
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
    pCommCD->remSIdTag = smSIdTag;
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->iCommMode = ShaComm_ASKJOIN;
    setupCollabAskJoinCommDialog(pCommCD);
    setCollabCommData(smSIdTag, pFrontSId->lSIDTag,
              ShaComm_ASKJOIN, pCommCD);
  }
  defaultDialogPopup(pCommCD->pDialogCD);
}

void
collabDeleteAskJoinPanelOprn(smSIdTag)
     shastraIdTag smSIdTag;
{
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                  ShaComm_ASKJOIN);
  if(pCommCD != NULL){
    freeCollabCommData(smSIdTag, pFrontSId->lSIDTag, ShaComm_ASKJOIN);
    free(pCommCD);
  }
}


/*
 * Function
 */
```

```
void
collabTellJoinOprn(smSIdTag, sIdTag)
     shastraIdTag    smSIdTag;
     shastraIdTag    sIdTag;
{
  if (frontIsInCollSession(sIdTag, smSIdTag)) {
    collabUtilPopupMessage("Already in this Session!\n");
    return;
  }
  if(collTellJoinReq(pHostKernel, &smSIdTag, &sIdTag,
             (shastraIdTag *) & collPermissions) == -1){
    collabUtilPopupMessage("collTellJoinReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabAskJoinPromptOprn(smSIdTag, fromSIdTag)
     shastraIdTag smSIdTag, fromSIdTag;
{
  shastraIdTags *pSIdTags;
  shastraId *pSesmSId, *pRemSId;
  shaCommCntlData *pCommCD;
  unsigned long lRespStatus;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Ask-Join Session!\n");
    return;
  }
  pRemSId = krFrSIdTag2SId(fromSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Join Requestor!\n");
    return;
  }
  if(frontIsInCollSession(fromSIdTag, smSIdTag)){
    collabUtilPopupMessage("Requestor Already in Session!\n");
    return;
  }

  pSIdTags = getSesmFrontSIdTags(&smSIdTag);

  switch(collOptionState.iAskJoinOpt){
  case CollabOpt_ALLOW:
    collabUtilPopupMessage("Automatically Allowed Session Join!\n");
    lRespStatus = 1;
    if(collAskJnStatusReq(pHostKernel, &smSIdTag, &fromSIdTag,
              &pFrontSId->lSIDTag, lRespStatus) == -1){
      collabUtilPopupMessage("collAskJnStatusReq() Error!\n");
      return;
```

```
    }
    collabTellJoinOprn(smSIdTag, fromSIdTag);
    return;
    break;
  case CollabOpt_DENY:
    collabUtilPopupMessage("Automatically Denied Session Join!\n");
    lRespStatus = 0;
    if(collAskJnStatusReq(pHostKernel, &smSIdTag, &fromSIdTag,
              &pFrontSId->lSIDTag, lRespStatus) == -1){
      collabUtilPopupMessage("collAskJnStatusReq() Error!\n");
      return;
    }
    return;
    break;
  default:
    break;
  }

  if((pCommCD = getCollabCommData(smSIdTag, fromSIdTag, ShaComm_ASKJNRESP))
     == NULL){
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
    pCommCD->remSIdTag = fromSIdTag;
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->iCommMode = ShaComm_ASKJNRESP;
    setupCollabAskJnRespCommDialog(pCommCD);
    setCollabCommData(smSIdTag, fromSIdTag, ShaComm_ASKJNRESP, pCommCD);
  }
  /*got another askjoin from same tool for same conference.. ignore??*/
  defaultDialogPopup(pCommCD->pDialogCD);
}

/*
 * Function
 */
void
collabSetAskJoinStatusOprn(smSIdTag, toSIdTag, lStatus)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
     unsigned long lStatus;
{
  shastraId *pSesmSId;
  shastraId *pRemSId;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Join Session!\n");
    return;
  }
  pRemSId = krFrSIdTag2SId(toSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Join Requester!\n");
```

```
    return;
  }
  if(collAskJnStatusReq(pHostKernel, &smSIdTag, &toSIdTag,
            &pFrontSId->lSIDTag, lStatus) == -1){
    collabUtilPopupMessage("collAskJnStatusReq() Error!\n");
    return;
  }
}

void
collabAskJoinAllowOprn(smSIdTag, toSIdTag)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
{
  unsigned long lStatus = 1;
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, toSIdTag, ShaComm_ASKJNRESP);
  if(pCommCD != NULL){
    collabSetAskJoinStatusOprn(smSIdTag, toSIdTag, lStatus);
    freeCollabCommData(smSIdTag, toSIdTag, ShaComm_ASKJNRESP);
    free(pCommCD);
  }
  collabTellJoinOprn(smSIdTag, toSIdTag);
}

void
collabAskJoinDenyOprn(smSIdTag, toSIdTag)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
{
  unsigned long lStatus = 0;
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, toSIdTag, ShaComm_ASKJNRESP);
  if(pCommCD != NULL){
    collabSetAskJoinStatusOprn(smSIdTag, toSIdTag, lStatus);
    freeCollabCommData(smSIdTag, toSIdTag, ShaComm_ASKJNRESP);
    free(pCommCD);
  }
}

/*
 * Function
 */
void
collabShowAskJoinStatusOprn(smSIdTag, toSIdTag, sIdTag, lStatus)
     shastraIdTag smSIdTag,toSIdTag,sIdTag;
     unsigned long lStatus;
{
  char msgBuf[256];
  char *sName;
  shaCommCntlData *pCommCD;
```

```
  sName = mapSIdTag2Str(&sIdTag, PSIDNMHOST | PSIDNMAPPL | PSIDNMUSER);

  if(lStatus){
    sprintf(msgBuf,"(%s)\n has allowed participation\n", sName);
  }
  else{
    sprintf(msgBuf,"(%s)\n has denied participation\n", sName);
  }
  free(sName);
  if((pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                 ShaComm_ASKJOIN)) != NULL){
    shastraCommAppendText(pCommCD, msgBuf);
  }
  else{
    collabUtilPopupMessage(msgBuf);
  }
}

void
collabSendAskJoinMessageOprn(smSIdTag, msg)
     shastraIdTag smSIdTag;
     char *msg;
{
  shastraId *pSesmSId;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Join Session!\n");
    return;
  }
  if(collAskJoinMsgReq(pHostKernel, &smSIdTag,
              &pFrontSId->lSIDTag, msg) == -1){
    collabUtilPopupMessage("collAskJoinMsgReq() Error!\n");
    return;
  }
}

void
collabRecvdAskJoinMessageOprn(smSIdTag, fromSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag fromSIdTag;
     char *msg;
{
  shaCommCntlData *pCommCD;

  if((pCommCD = getCollabCommData(smSIdTag, fromSIdTag, ShaComm_ASKJNRESP))
     != NULL){
    /*should've been prompted, so if no panel, commited*/
    shastraCommDisplayText(pCommCD, msg);
  }
  else if(fromSIdTag == pFrontSId->lSIDTag){ /*joined empty collab*/
    pCommCD = getCollabCommData(smSIdTag, fromSIdTag, ShaComm_ASKJOIN);
```

```
    if(pCommCD != NULL){
      shastraCommDisplayText(pCommCD, msg);
/*should we terminate*/
    }
    else{
      collabUtilPopupMessage("You're the Session Leader!\n");
    }
  }
}

void
collabSendAskJnRespMessageOprn(smSIdTag, toSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag toSIdTag;
     char *msg;
{
  shastraId *pSesmSId;
  shastraId *pRemSId;

  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
    collabUtilPopupMessage("Invalid Join Session!\n");
    return;
  }
  pRemSId = krFrSIdTag2SId(toSIdTag);
  if(pRemSId == NULL){
    collabUtilPopupMessage("Invalid Requestor for Join!\n");
    return;
  }
  if(collAskJnRespMsgReq(pHostKernel, &smSIdTag, &toSIdTag,
             &pFrontSId->lSIDTag, msg) == -1){
    collabUtilPopupMessage("collAskJnRespMsgReq() Error!\n");
    return;
  }
}

void
collabRecvdAskJnRespMessageOprn(smSIdTag, fromSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag fromSIdTag;
     char *msg;
{
  shaCommCntlData *pCommCD;

  if((pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                 ShaComm_ASKJOIN)) != NULL){
    shastraCommDisplayText(pCommCD, msg);
  }
}

/*
 * Function
 */
```

```
void
collabRemoveOprn(i)
     int i;
{
  shastraId      *pSesmSId;
  shastraIdTag   *pSIdTag;
  shastraIdTags  *pSIdTags;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Remove!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  if ((i < 0) || (i >= pSIdTags->shastraIdTags_len)) {
    collabUtilPopupMessage("System not in Current Session!\n");
    return;
  }
  pSIdTag = &pSIdTags->shastraIdTags_val[i];

  if(collRemoveReq(pHostShaCurrColl, pSIdTag) == -1){
    collabUtilPopupMessage("collRemoveReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabCommConnectOprn(i)
     int i;
{
  shastraId      *pSesmSId;
  shastraId      *pRemSId;
  shastraIdTag   lSIdTag;
  shastraIdTags  *pSIdTags;
  shaCommCntlData *pCommCD;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  if ((i < 0) || (i >= pSIdTags->shastraIdTags_len)) {
    collabUtilPopupMessage("System not in Current Session!\n");
```

```
      return;
    }
    lSIdTag = pSIdTags->shastraIdTags_val[i];

    if(lSIdTag == pFrontSId->lSIDTag){
      /*wasteful, so disallow*/
    }
    if((pCommCD = getCollabCommData(currCollSIdTag, lSIdTag, ShaComm_COLLAB))
       == NULL){
      pRemSId = krFrSIdTag2SId(lSIdTag);
      if(pRemSId == NULL){
        collabUtilPopupMessage("Couldn't Locate Remote System!\n");
        return;
      }
      pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
      memset(pCommCD, 0, sizeof(shaCommCntlData));
      pCommCD->locSIdTag = pFrontSId->lSIDTag;
      pCommCD->remSIdTag = lSIdTag;
      pCommCD->smSIdTag = currCollSIdTag;
      pCommCD->iCommMode = ShaComm_COLLAB;
      setupCollabCommDialog(pCommCD);
      setCollabCommData(currCollSIdTag, lSIdTag, ShaComm_COLLAB, pCommCD);
      collabCommSendMessageOprn(currCollSIdTag, pRemSId->lSIDTag, "");
      /*force remote popup*/
    }
    defaultDialogPopup(pCommCD->pDialogCD);
}

void
collabCommSendMessageOprn(smSIdTag, lSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag lSIdTag;
     char *msg;
{
    if(collCommMsgTextReq(pHostShaCurrColl, &smSIdTag, &lSIdTag,
              &pFrontSId->lSIDTag, msg) == -1){
      collabUtilPopupMessage("collCommMsgTextReq() Error!\n");
      return;
    }
}

void
collabCommRecvdMessageOprn(smSIdTag, lSIdTag, msg)
     shastraIdTag smSIdTag;
     shastraIdTag lSIdTag;
     char *msg;
{
    shastraId      *pSesmSId;
    shastraId      *pRemSId;
    shaCommCntlData *pCommCD;

    pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
    if(pSesmSId == NULL){
```

```
    return;
  }
  if((pCommCD = getCollabCommData(smSIdTag, lSIdTag, ShaComm_COLLAB))
     == NULL){
    pRemSId = krFrSIdTag2SId(lSIdTag);
    if(pRemSId == NULL){
      collabUtilPopupMessage("Couldn't Locate Remote System!\n");
      return;
    }
    pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
    memset(pCommCD, 0, sizeof(shaCommCntlData));
    pCommCD->locSIdTag = pFrontSId->lSIDTag;
    pCommCD->remSIdTag = lSIdTag;
    pCommCD->smSIdTag = smSIdTag;
    pCommCD->iCommMode = ShaComm_COLLAB;
    setupCollabCommDialog(pCommCD);
    setCollabCommData(smSIdTag, lSIdTag, ShaComm_COLLAB, pCommCD);
  }
  shastraCommDisplayText(pCommCD, msg);
}

void
collabCommDisconnectOprn(smSIdTag, remSIdTag)
     shastraIdTag smSIdTag;
     shastraIdTag remSIdTag;
{
  shaCommCntlData *pCommCD;

  pCommCD = getCollabCommData(smSIdTag, remSIdTag, ShaComm_COLLAB);
  if(pCommCD != NULL){
    freeCollabCommData(smSIdTag, remSIdTag, ShaComm_COLLAB);
    free(pCommCD);
  }
}


void
collabOperationsOprn(pMgrCD, fUp)
     mgrCntlData *pMgrCD;
     int fUp;
{
  if(pHostShaCurrColl == NULL){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  if (collabOperatorFunc != NULL) {
    (*collabOperatorFunc) (pHostShaCurrColl);
  }
}

/*
 * Function
 */
```

```
void
collabSetPermOprn(iPerm, fSet)
     CollabPermission iPerm;
     int fSet;
{
  switch(iPerm){
  case CollabPerm_ACCESS:
    if(fSet){
      collPermissions |= SHASTRA_PERM_ACCESS;
    }
    else{
      collPermissions &= ~SHASTRA_PERM_ACCESS;
    }
    break;
  case CollabPerm_BROWSE:
    if(fSet){
      collPermissions |= SHASTRA_PERM_BROWSE;
    }
    else{
      collPermissions &= ~SHASTRA_PERM_BROWSE;
    }
    break;
  case CollabPerm_MODIFY:
    if(fSet){
      collPermissions |= SHASTRA_PERM_MODIFY;
    }
    else{
      collPermissions &= ~SHASTRA_PERM_MODIFY;
    }
    break;
  case CollabPerm_GRANT:
    if(fSet){
      collPermissions |= SHASTRA_PERM_GRANT;
    }
    else{
      collPermissions &= ~SHASTRA_PERM_GRANT;
    }
    break;
  case CollabPerm_COPY:
    if(fSet){
      collPermissions |= SHASTRA_PERM_COPY;
    }
    else{
      collPermissions &= ~SHASTRA_PERM_COPY;
    }
    break;
  }
}

/*
 * Function
 */
void
```

```
collabSetIxnModeOprn(iMode)
     CollabIxnMode iMode;
{
  shastraId      *pSesmSId;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Set Mode!\n");
    return;
  }
  if(iMode == CollabIxn_REGULATED){
    collIxnMode = SHASTRA_MODE_REGUL;
  }
  else{
    collIxnMode = SHASTRA_MODE_UNREG;
  }
  if(collSetIxnModeReq(pHostShaCurrColl, collIxnMode) == -1){
    collabUtilPopupMessage("collSetIxnModeReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabPermSetOprn(aiWhich)
     int *aiWhich;
{
  shastraId      *pSesmSId;
  shastraIdTag   *pSIdTag;
  shastraIdTags  *pSIdTags;
  int i;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Set Permissions!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  for(i = 0; i < pSIdTags->shastraIdTags_len; i++){
```

```
    if(aiWhich[i]){
      pSIdTag = &pSIdTags->shastraIdTags_val[i];
      if(collSetPermsReq(pHostShaCurrColl, pSIdTag,
             collPermissions) == -1){
    collabUtilPopupMessage("collSetPermsReq() Error!\n");
    return;
      }
    }
  }
}

/*
 * Function
 */
void
collabPermCheckOprn(iWhich)
     int iWhich;
{
  shastraId      *pSesmSId;
  shastraIdTag   *pSIdTag;
  shastraIdTags  *pSIdTags;
  unsigned long   perms;
  char msgBuf[256], *sName, *sPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  if ((iWhich < 0) || (iWhich >= pSIdTags->shastraIdTags_len)) {
    collabUtilPopupMessage("System not in Current Session!\n");
    return;
  }
  pSIdTag = &pSIdTags->shastraIdTags_val[iWhich];
  perms = getSesmFrontPerms(&currCollSIdTag, pSIdTag);

  sName = mapSIdTag2Str(pSIdTag, PSIDNMHOST | PSIDNMAPPL | PSIDNMUSER);
  sPerms = perms2Str(perms);
  sprintf(msgBuf,"%s has %s\n", sName, sPerms);
  free(sName); free(sPerms);
  collabUtilPopupMessage(msgBuf);

  if(*pSIdTag == pFrontSId->lSIDTag){
    setCollabFrontPermsOprn(currCollSIdTag);
  }
}

/*
 * Function
 */
static textDlgCntlData *pCollabTextDCD;
void
```

```
collabDescribeOprn(iWhich)
     int iWhich;
{
  shastraId      *pSesmSId, *pSId;
  shastraIdTag   *pSIdTag;
  shastraIdTags  *pSIdTags;
  char *str, *sPerms, msgBuf[128];
  unsigned long   perms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  if ((iWhich < 0) || (iWhich >= pSIdTags->shastraIdTags_len)) {
    collabUtilPopupMessage("System not in Current Session!\n");
    return;
  }
  pSIdTag = &pSIdTags->shastraIdTags_val[iWhich];
  pSId = mapSIdTag2SId(pSIdTag);
  if(pSId == NULL){
    collabUtilPopupMessage("Invalid System!\n");
    return;
  }
  if(pCollabTextDCD == NULL){
    pCollabTextDCD = (textDlgCntlData*)malloc(sizeof(textDlgCntlData));
    memset(pCollabTextDCD, 0, sizeof(textDlgCntlData));
    pCollabTextDCD->sName = "Shastra Description";
    pCollabTextDCD->fnDestroyCallback = collabShowTextDestroyOprn;
    pCollabTextDCD->fBehave = DIALOG_AUTOLOWER;
    pCollabTextDCD->iDelay = 60000;
    setupTextDialog(pFrontAppData->wgTop, pCollabTextDCD, NULL);
  }

  str = pSId2StrDetail(pSId, 0);
  textDialogAppendText(pCollabTextDCD, str);
  free(str);
  perms = getSesmFrontPerms(&currCollSIdTag, pSIdTag);
  sPerms = perms2Str(perms);
  sprintf(msgBuf,"Permissions    : %s\n", sPerms);
  free(sPerms);
  textDialogAppendText(pCollabTextDCD, msgBuf);
}

void
collabShowTextDestroyOprn(pTextCD)
     textDlgCntlData *pTextCD;
{
  if(pTextCD != pCollabTextDCD){
    return;
  }
  if(pCollabTextDCD != NULL){
```

```
    free(pCollabTextDCD);
    pCollabTextDCD = NULL;
  }
}

/*
 * Function
 */
void
collabFloorSetOprn(iWhich)
     int iWhich;
{
  shastraId      *pSesmSId;
  shastraIdTag   *pSIdTag;
  shastraIdTags  *pSIdTags;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Set Floor!\n");
    return;
  }
  pSIdTags = getSesmFrontSIdTags(&currCollSIdTag);
  if ((iWhich < 0) || (iWhich >= pSIdTags->shastraIdTags_len)) {
    collabUtilPopupMessage("System not in Current Session!\n");
    return;
  }
  pSIdTag = &pSIdTags->shastraIdTags_val[iWhich];
  if(collTellTokenReq(pHostShaCurrColl, pSIdTag) == -1){
    collabUtilPopupMessage("collTellTokenReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabSetSesFormatOprn(iMode)
     CollabFmtMode iMode;
{
  shastraId      *pSesmSId;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
```

```
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Set Format!\n");
    return;
  }
  if(iMode == CollabFmt_FORMAL){
    collFormat = SHASTRA_FORMAT_FORMAL;
  }
  else{
    collFormat = SHASTRA_FORMAT_INFORMAL;
  }
  if(collSetSesFormatReq(pHostShaCurrColl, collFormat) == -1){
    collabUtilPopupMessage("collSetSesFormatReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabSetFloorModeOprn(iMode)
     CollabFlrMode iMode;
{
  shastraId      *pSesmSId;
  unsigned long   myPerms;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  myPerms = getSesmFrontPerms(&currCollSIdTag, & pFrontSId->lSIDTag);
  if (!(myPerms & SHASTRA_PERM_GRANT)) {
    collabUtilPopupMessage("No Capability to Set Format!\n");
    return;
  }
  switch(iMode){
  case CollabFlr_MODERATED:
    collFloorMode = SHASTRA_FLOOR_MODERATED;
    break;
  case CollabFlr_VOLUNTARY:
    collFloorMode = SHASTRA_FLOOR_VOLUNTARY;
    break;
  case CollabFlr_CYCLIC:
    collFloorMode = SHASTRA_FLOOR_CYCLIC;
    break;
  case CollabFlr_PREEMPTIVE:
    collFloorMode = SHASTRA_FLOOR_PREEMPTIVE;
    break;
  case CollabFlr_HANDOFF:
    collFloorMode = SHASTRA_FLOOR_HANDOFF;
```

```
    break;
  case CollabFlr_QUEUED:
    collFloorMode = SHASTRA_FLOOR_QUEUED;
    break;
  default:
    return;
  }
  if(collSetFloorModeReq(pHostShaCurrColl, collFloorMode) == -1){
    collabUtilPopupMessage("collSetFloorModeReq() Error!\n");
    return;
  }
}

/*
 * Function
 */
void
collabFreeFloorOprn(fSet)
     int fSet;
{
  shastraId      *pSesmSId;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  if(sIdTagToken == pFrontSId->lSIDTag){
    if(collFreeTokenReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collFreeTokenReq() Error!\n");
      return;
    }
  }
  fFreeFloor = fSet;
}

/*
 * Function
 */
void
collabRequestFloorOprn()
{
  shastraId      *pSesmSId;

  pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
  if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
    collabUtilPopupMessage("Invalid Current Session!\n");
    return;
  }
  if(sIdTagToken != pFrontSId->lSIDTag){
    if(collGrabTokenReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collGrabTokenReq() Error!\n");
      return;
```

```
    }
  }
}

void
collabSetInviteOption(iCollOpt)
     CollabOption iCollOpt;
{
  collOptionState.iInviteOpt = iCollOpt;
}

void
collabSetAskJoinOption(iCollOpt)
     CollabOption iCollOpt;
{
  collOptionState.iAskJoinOpt = iCollOpt;
}

void
collabSetStartOption(iCollOpt)
     CollabOption iCollOpt;
{
  collOptionState.iStartOpt = iCollOpt;
}

void
collabSetForceJoinOption(iCollOpt)
     CollabOption iCollOpt;
{
  collOptionState.iForceJoinOpt = iCollOpt;
}

void
collabSyncOprn(iCmd)
     CollabCmd iCmd;
{
  shastraId      *pSesmSId;

  if(iCmd != CollabCmd_SYNCSESSION){
    pSesmSId = getSIdByTagInSIds(&currCollSIdTag, &shastraSesmIds);
    if((pSesmSId == NULL) || (pHostShaCurrColl == NULL)){
      collabUtilPopupMessage("Invalid Current Session!\n");
      return;
    }
  }
  switch(iCmd){
  case CollabCmd_SYNCSESSION:
    if(getShaSesmIdReq(pHostKernel) == -1){
      collabUtilPopupMessage("getShaSesmIdReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCSESSFR:
```

```
    if(getShaSesmFrIdReq(pHostKernel, &currCollSIdTag) == -1){
      collabUtilPopupMessage("getShaSesmFrIdReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCFORMAT:
    if(collGetSesFormatReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collGetSesFormatReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCIXNMODE:
    if(collGetIxnModeReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collGetIxnModeReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCFLRMODE:
    if(collGetFloorModeReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collGetFloorModeReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCFLOOR:
    if(collAskTokenReq(pHostShaCurrColl) == -1){
      collabUtilPopupMessage("collAskTokenReq() Error!\n");
      return;
    }
    break;
  case CollabCmd_SYNCPERMS:
    if(collGetPermsReq(pHostShaCurrColl,
               &pFrontSId->lSIDTag) == -1){
      collabUtilPopupMessage("collGetPermsReq() Error!\n");
      return;
    }
    break;
  }
}

/*
 * Function
 */
int
frontIsInCollSession(sIdTag, smSIdTag)
     shastraIdTag   sIdTag;
     shastraIdTag   smSIdTag;
{
  shastraIdTags  *pSIdTags;
  int             iFront;

  pSIdTags = getSesmFrontSIdTags(&smSIdTag);
  iFront = getSIdTagIndexInSIdTags(&sIdTag, pSIdTags);
  if (iFront < 0) {
```

```
    return 0;
  } else {
    return 1;
  }
}




#define MAPSIZE 64
static collCommMapEntry collMap[MAPSIZE];

shaCommCntlData *
getCollabCommData(smSIdTag, lSIdTag, iMode)
     shastraIdTag smSIdTag;
     shastraIdTag lSIdTag;
     ShastraCommMode iMode;
{
  int i;
  if(!smSIdTag || !lSIdTag || !iMode){
    return;
  }
  for(i=0; i < MAPSIZE;i++){
    if((smSIdTag == collMap[i].smSIdTag) &&
       (lSIdTag == collMap[i].lSIdTag) &&
       (iMode == collMap[i].iMode)){
      return collMap[i].pCommCD;
    }
  }
  return NULL;
}

int
setCollabCommData(smSIdTag, lSIdTag, iMode, pCommCD)
     shastraIdTag smSIdTag;
     shastraIdTag lSIdTag;
     ShastraCommMode iMode;
     shaCommCntlData *pCommCD;
{
  int i;
  if(!smSIdTag || !lSIdTag || !iMode || pCommCD == NULL){
    return -1;
  }
  for(i=0; i < MAPSIZE;i++){
    if(collMap[i].smSIdTag == 0){
      collMap[i].smSIdTag = smSIdTag;
      collMap[i].lSIdTag = lSIdTag;
      collMap[i].iMode = iMode;
      collMap[i].pCommCD = pCommCD;
      return i;
    }
```

```
  }
  fprintf(stderr,"setCollabCommData(%ld,%ld,%d)->couldn't set!\n",
      smSIdTag, lSIdTag, iMode);
  return -1;
}

int
freeCollabCommData(smSIdTag, lSIdTag, iMode)
     shastraIdTag smSIdTag;
     shastraIdTag lSIdTag;
     ShastraCommMode iMode;
{
  int i;
  if(!smSIdTag || !lSIdTag || !iMode){
    return;
  }
  for(i=0; i < MAPSIZE;i++){
    if((smSIdTag == collMap[i].smSIdTag) &&
       (lSIdTag == collMap[i].lSIdTag) &&
       (iMode == collMap[i].iMode)){
      collMap[i].smSIdTag = 0;
      collMap[i].lSIdTag = 0;
      collMap[i].iMode = 0;
      collMap[i].pCommCD = NULL;
      return i;
    }
  }
  fprintf(stderr,"freeCollabCommData(%ld,%ld,%d)->couldn't free!\n",
      smSIdTag, lSIdTag, iMode);
  return -1;
}




void
collabAskJoinOprnSilent(int i)
{
  shastraIdTag smSIdTag;
  shaCommCntlData *pCommCD;
  shastraId *pSesmSId;

  if ((i < 0) || (i >= shastraSesmIds.shastraIds_len)){
    return;
  }
  smSIdTag = shastraSesmIds.shastraIds_val[i]->lSIDTag;
  if(frontIsInCollSession(pFrontSId->lSIDTag, smSIdTag)){
    collabUtilPopupMessage("System Already In Session!\n");
    return;
  }
  pSesmSId = getSIdByTagInSIds(&smSIdTag, &shastraSesmIds);
  if(pSesmSId == NULL){
```

```
      collabUtilPopupMessage("Invalid Ask-Join Session!\n");
      return;
    }
    if(collAskJoinReq(pHostKernel, &smSIdTag,
              &pFrontSId->lSIDTag) == -1){
      collabUtilPopupMessage("collAskJoinReq() Error!\n");
      return;
    }

    if((pCommCD = getCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                    ShaComm_ASKJOIN)) == NULL){
      pCommCD = (shaCommCntlData*)malloc(sizeof(shaCommCntlData));
      memset(pCommCD, 0, sizeof(shaCommCntlData));
      pCommCD->locSIdTag = pFrontSId->lSIDTag;
      pCommCD->remSIdTag = smSIdTag;
      pCommCD->smSIdTag = smSIdTag;
      pCommCD->iCommMode = ShaComm_ASKJOIN;
      setupCollabAskJoinCommDialog(pCommCD);
      setCollabCommData(smSIdTag, pFrontSId->lSIDTag,
                ShaComm_ASKJOIN, pCommCD);
    }
    /*defaultDialogPopup(pCommCD->pDialogCD);*/
  }
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/*
 * collabCntlUI.c
 */
#include <stdio.h>

#include <Xm/CascadeB.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleB.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/PushB.h>
#include <Xm/Label.h>
#include <Xm/Text.h>
#include <Xm/SelectioB.h>
#include <Xm/Separator.h>
#include <Xm/Xm.h>
#include <X11/Shell.h>

#include <shastra/uitools/chooseMany.h>
#include <shastra/uitools/chooseOne.h>
#include <shastra/uitools/menu.h>
#include <shastra/uitools/toggles.h>
#include <shastra/uitools/buttons.h>
#include <shastra/uitools/genui.h>
#include <shastra/uitools/dialog.h>
#include <shastra/uitools/text.h>
#include <shastra/uitools/choose.h>
#include <shastra/uitools/controlPanel.h>
```

```
#include <shastra/shastra.h>
#include <shastra/front/front.h>
#include <shastra/front/frontP.h>
#include <shastra/front/frontState.h>
#include <shastra/front/collabCntl.h>
#include <shastra/front/collabCntlP.h>
#include <shastra/front/shastraCntlP.h>
#include <shastra/front/shastraCntl.h>

Widget createHelpPD();

static void collabOptionCB();
static void collabOperationCB();
static void collabDismissCB();
static void collabCreateBrowserCB();
static void collabShowTraceCB();
static void collabSyncCB();
static void collabDbgSendCB();
static void collabDbgCheckCB();
static void collabFreeFloorCB();
static void collabSetIxnModeCB();
static void collabCmdCB();
static void collabSetSesFormatCB();
static void collabSetFloorModeCB();
static void collabFreeFloorCB();
static void collabRequestFloorCB();
static void collabFloorParamsCB();
static void chooseOneCollabCB();
static void createCollabCntlAreaCB();
static void collabInitiateCMCB();
static void collabInviteCMCB();
static void collabJoinCOCB();
static void collabRemoveCOCB();
static void collabConnectCOCB();
static void collabPermSetCMCB();
static void collabPermCheckCOCB();
static void collabDescribeCOCB();
static void collabFloorSetCOCB();
static void collabSetPermsDialogPopup();
static void collabCommDismissCB();
static void collabCommTerminateCB();
static void collabCommClearCB();
static void collabCommTextCB();
static void collabCommDismissCB();
static void collabCommTerminateCB();
static void collabInviteCommDismissCB();
static void collabInviteCommClearCB();
static void collabInviteCommTerminateCB();
static void collabInvRespCommClearCB();
static void collabInvRespCommAcceptCB(), collabInvRespCommDeclineCB();
static void collabInvRespCommTextCB();
static void collabInvRespCommAcceptCB();
static void collabInvRespCommDeclineCB();
```

```
static void collabAskJnRespCommClearCB();
static void collabAskJnRespCommAllowCB();
static void collabAskJnRespCommDenyCB();
static void collabAskJnRespCommTextCB();
static void collabAskJnRespCommAllowCB();
static void collabAskJnRespCommDenyCB();
static void collabAskJoinCommDismissCB();
static void collabAskJoinCommClearCB();
static void collabAskJoinCommTerminateCB();
static void collabSetPermModeCB();
static void collabPermsCB();
static void collabSetPermModeCB();
static void collabSetPermsCB();
static void collabAskJoinCommTextCB();
static void collabAskJoinCommDismissCB();
static void collabAskJoinCommTerminateCB();
static Widget createCollAskJnRespCommMenuBar();
static Widget createCollInvRespCommMenuBar();
static Widget createCollCommMenuBar();
static Widget createCollAskJoinCommMenuBar();


static void systemGenChooseManySetup(Prot2(Widget, multiChooseCntlData*));
static void systemGenChooseManyCB(Prot3(Widget, XtPointer, XtPointer));
static void systemGenChooseOneSetup(Prot2(Widget, optChooseCntlData*));
static void systemGenChooseOneCB(Prot3(Widget, XtPointer, XtPointer));

static void collabGenChooseOneSetup(Prot2(Widget, optChooseCntlData*));
static void collabGenChooseOneCB(Prot3(Widget, XtPointer, XtPointer));

static void sesMgrGenChooseOneSetup(Prot2(Widget, optChooseCntlData*));
static void sesMgrGenChooseOneCB(Prot3(Widget, XtPointer, XtPointer));

static void collabFrontGenChooseOneSetup(Prot2(Widget, optChooseCntlData*))
    ;
static void collabFrontGenChooseOneCB(Prot3(Widget, XtPointer, XtPointer));
static void collabFrontGenChooseManySetup(Prot2(Widget, multiChooseCntlData
    *));
static void collabFrontGenChooseManyCB(Prot3(Widget, XtPointer, XtPointer))
    ;


chooseOne     *pcoCollab;
chooseOne     *pcoCollabFronts;
chooseMany    *pcmCollabFronts;

static chooseOne     *pcoSystems;
static chooseMany    *pcmSystems;
static chooseOne     *pcoSesMgrs;

static int fDebugTrace = 1;
static mgrCntlData *pCollabDismissData;
static textCntlData collabMsgBufCntl = {"collMsgBuffer", NULL, NULL};
```

```
static Widget wgCollabShell;

static Widget
createCollabMenuBar(wgParent, sName, argList)
     Widget          wgParent;
     char           *sName;
     XtVarArgsList   argList;
{
  Widget          wgMenuBar;
  Widget          wgControlPD, wgOptionPD, wgDebugPD, wgHelpPD;
  Arg             args[8];
  int             n;
  static mgrCntlData cntlOperation;
  static mgrCntlData collBrowser;
  static menuItem controlPD[] = {
    {"App. Operations", (XtPointer) &cntlOperation, False,
        collabOperationCB},
    {"Session Browser", (XtPointer)&collBrowser, False,
        collabCreateBrowserCB},
    {"sep", (XtPointer) NULL, False, NULL, NULL, &xmSeparatorWidgetClass},
    {"Dismiss", (XtPointer) NULL, False, collabDismissCB},
    {NULL}
  };
  static menuItem syncPD[] = {
    {"Sessions", (XtPointer)CollabCmd_SYNCSESSION, False, collabSyncCB},
    {"Session Fronts", (XtPointer)CollabCmd_SYNCSESSFR, False, collabSyncCB
        },
    {"Session Format", (XtPointer)CollabCmd_SYNCFORMAT, False, collabSyncCB
        },
    {"Interaction Mode", (XtPointer)CollabCmd_SYNCIXNMODE, False,
        collabSyncCB},
    {"Floor Mode", (XtPointer)CollabCmd_SYNCFLRMODE, False, collabSyncCB},
    {"Floor", (XtPointer)CollabCmd_SYNCFLOOR, False, collabSyncCB},
    {"Permissions", (XtPointer)CollabCmd_SYNCPERMS, False, collabSyncCB},
    {NULL}
  };
  static menuItem debugPD[] = {
    {"Sync.", NULL, False, NULL, NULL, NULL, syncPD, MENU_PUSH},
    {"Trace", (XtPointer)NULL, True, collabShowTraceCB, NULL, &
        xmToggleButtonWidgetClass},
    {NULL}
  };
/*
  {"Data Send", (XtPointer)NULL, False, collabDbgSendCB },
  {"Data Check", (XtPointer)NULL, False, collabDbgCheckCB },
  */
  static menuItem optionPD[] = {
    {"Auto-Accept", (XtPointer) CollabOpt_ACCEPT, False, collabOptionCB},
    {"Auto-Decline", (XtPointer) CollabOpt_DECLINE, False, collabOptionCB},
    {"sep", (XtPointer)NULL, False, NULL, NULL, &xmSeparatorWidgetClass},
    {"Auto-Allow", (XtPointer) CollabOpt_ALLOW, False, collabOptionCB},
    {"Auto-Deny", (XtPointer) CollabOpt_DENY, False, collabOptionCB},
    {"sep", (XtPointer)NULL, False, NULL, NULL, &xmSeparatorWidgetClass},
```

```
    {"Auto-Start", (XtPointer) CollabOpt_START, False, collabOptionCB},
    {"sep", (XtPointer)NULL, False, NULL, NULL, &xmSeparatorWidgetClass},
    {"Force-Join", (XtPointer) CollabOpt_FORCEJOIN, False, collabOptionCB},
    {NULL}
  };

  n = 0;
  if (argList) {
    XtSetArg(args[n], XtVaNestedList, argList);
    n++;
  }
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;

  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                  controlPD, NULL);

  wgOptionPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                 optionPD, NULL);
  wgDebugPD = pulldownMenuCreate(wgMenuBar, "Debug", MENU_MIXED,
                debugPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}

static toggleItem collabPermTgls[] = {
  {"Access",(XtPointer)CollabPerm_ACCESS, False, NULL},
  {"Browse",(XtPointer)CollabPerm_BROWSE, False, NULL},
  {"Modify",(XtPointer)CollabPerm_MODIFY, False, NULL},
  {"Copy",(XtPointer)CollabPerm_COPY, False, NULL},
  {"Grant",(XtPointer)CollabPerm_GRANT, False, NULL},
  {NULL}
};
void
collabSetPermToggles(lPerms)
     unsigned long lPerms;
{
  if(collabPermTgls[0].wgTgl == NULL){
    return;
  }
  togglesSetState(collabPermTgls, (XtPointer)CollabPerm_ACCESS,
          (lPerms & SHASTRA_PERM_ACCESS), True);
  togglesSetState(collabPermTgls, (XtPointer)CollabPerm_BROWSE,
          (lPerms & SHASTRA_PERM_BROWSE), True);
  togglesSetState(collabPermTgls, (XtPointer)CollabPerm_MODIFY,
          (lPerms & SHASTRA_PERM_MODIFY), True);
  togglesSetState(collabPermTgls, (XtPointer)CollabPerm_COPY,
          (lPerms & SHASTRA_PERM_COPY), True);
```

```
  togglesSetState(collabPermTgls, (XtPointer)CollabPerm_GRANT,
          (lPerms & SHASTRA_PERM_GRANT), True);
}

static toggleItem collabFloorTgls[] = {
  {"Have It?",(XtPointer)CollabCmd_HAVEFLOOR, False, NULL},
  {"Release?",(XtPointer)CollabCmd_FREEFLOOR, False, collabFreeFloorCB},
  {NULL}
};
static Widget wgCollabFloorLabel;
void
collabSetFloorInfo(sName, fHave, fFree)
     char *sName;
     int fHave, fFree;
{
  XmString str;

  if(wgCollabFloorLabel == NULL){
    return;
  }
  if(sName == NULL){
    sName = "<Leader>";
  }
  str = XmStringCreateSimple(sName);
  XtVaSetValues(wgCollabFloorLabel, XmNlabelString, str, NULL);
  XmStringFree(str);
  togglesSetState(collabFloorTgls, (XtPointer)CollabCmd_HAVEFLOOR,
          fHave, True);
  togglesSetState(collabFloorTgls, (XtPointer)CollabCmd_FREEFLOOR,
          fFree, True);
}

static void
createCollabCntlAreaCB(wgParent, xpClient, xpCall)
     Widget          wgParent;
     XtPointer       xpClient, xpCall;
{
  Widget wgDbgText, wgIxnForm, wgFlrTitle;
  Widget wgListRC, wgPermTglRC, wgOMRC, wgFloorRC, wgFlrTglRC, wgFlrCmdRC;
  Widget wgModeOM, wgFloorOM, wgFormatOM, wgFlrFrame;
  Arg             args[16];
  int             i, n;
  XmString str;
  XtVarArgsList   argList;
  static buttonItem flrCmdBtns[] = {
    {"Request Floor",(XtPointer)NULL, collabRequestFloorCB},
    {"Parameters",(XtPointer)NULL, collabFloorParamsCB},
    {NULL}
  };
  static menuItem modeOM[] = {
    {"Unregulated", (XtPointer) CollabIxn_UNREGULATED, False,
        collabSetIxnModeCB},
    {"Regulated", (XtPointer) CollabIxn_REGULATED, False,
```

```
        collabSetIxnModeCB},
    {NULL}
};
static menuItem formatOM[] = {
    {"Formal     ", (XtPointer) CollabFmt_FORMAL, False,
        collabSetSesFormatCB},
    {"Informal", (XtPointer) CollabFmt_INFORMAL, False,
        collabSetSesFormatCB},
    {NULL}
};
static menuItem floorOM[] = {
    {"Moderated  ", (XtPointer) CollabFlr_MODERATED, False,
        collabSetFloorModeCB},
    {"Handoff", (XtPointer) CollabFlr_HANDOFF, False, collabSetFloorModeCB}
        ,
    {"Cyclic", (XtPointer) CollabFlr_CYCLIC, False, collabSetFloorModeCB},
    {"Queued", (XtPointer) CollabFlr_QUEUED, False, collabSetFloorModeCB},
    {"Voluntary", (XtPointer) CollabFlr_VOLUNTARY, False,
        collabSetFloorModeCB},
    {"Preemptive", (XtPointer) CollabFlr_PREEMPTIVE, False,
        collabSetFloorModeCB},
    {NULL}
};
static buttonItem panelBtns[] = {
    {"  Initiate ",(XtPointer)CollabCmd_INITIATE, collabCmdCB},
    {"Invite",(XtPointer)CollabCmd_INVITE, collabCmdCB},
    {"Join",(XtPointer)CollabCmd_JOIN, collabCmdCB},
    {"Remove",(XtPointer)CollabCmd_REMOVE, collabCmdCB},
    {"Set Perms",(XtPointer)CollabCmd_PERMSET, collabCmdCB},
    {"Check Perms",(XtPointer)CollabCmd_PERMCHK, collabCmdCB},
    {"Set Floor",(XtPointer)CollabCmd_FLOOR, collabCmdCB},
    {"Describe",(XtPointer)CollabCmd_DESCRIBE, collabCmdCB},
    {"Converse",(XtPointer)CollabCmd_CONNECT, collabCmdCB},
    {NULL}
};
char *sXlns = "#override \n <Btn1Up>: \n <Btn1Down> :\n";
XtTranslations pXlns;


wgIxnForm = XtVaCreateWidget("collabIxnForm", xmFormWidgetClass,
                wgParent,
                XmNtopAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_FORM,
                XmNrightAttachment, XmATTACH_FORM,
                NULL);

argList = XtVaCreateArgsList(NULL,
                XmNleftAttachment, XmATTACH_FORM,
                XmNtopAttachment, XmATTACH_FORM,
                NULL);
wgListRC = buttonsCreate(wgIxnForm, "panelCmds", XmVERTICAL,
             panelBtns, argList);
XtManageChild(wgListRC);
```

```
    XtFree(argList);

    wgOMRC = XtVaCreateWidget("modeOMRC", xmRowColumnWidgetClass,
                wgIxnForm,
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, wgListRC,
                XmNtopAttachment, XmATTACH_FORM,
                XmNrightAttachment, XmATTACH_FORM,
                NULL);
    wgFormatOM = optionMenuCreate(wgOMRC, "Session Format  ", MENU_PUSH,
                XmHORIZONTAL, formatOM, NULL);
    wgModeOM = optionMenuCreate(wgOMRC, "Interaction Mode", MENU_PUSH,
                  XmHORIZONTAL, modeOM, NULL);
    wgFloorOM = optionMenuCreate(wgOMRC, "Floor Control   ", MENU_PUSH,
                   XmHORIZONTAL, floorOM, NULL);
    XtManageChild(wgOMRC);

    argList = XtVaCreateArgsList(NULL,
                   XmNleftAttachment, XmATTACH_WIDGET,
                   XmNleftWidget, wgListRC,
                   XmNtopAttachment, XmATTACH_WIDGET,
                   XmNtopWidget, wgOMRC,
                   NULL);
    wgPermTglRC = togglesCreate(wgIxnForm, "permTgls", TGL_CHECK, XmVERTICAL,
                  collabPermTgls, argList);
    pXlns = XtParseTranslationTable(sXlns);
    for(i=0; i<5; i++){
      XtOverrideTranslations(collabPermTgls[i].wgTgl, pXlns);
    }

    XtManageChild(wgPermTglRC);
    XtFree(argList);

    wgFlrFrame = XtVaCreateWidget("collabFlrFrame", xmFrameWidgetClass,
                wgIxnForm,
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, wgPermTglRC,
                XmNtopAttachment, XmATTACH_WIDGET,
                XmNtopWidget, wgOMRC,
                XmNrightAttachment, XmATTACH_FORM,
                XmNrightOffset, 5,
                NULL);
    wgFloorRC = XtVaCreateWidget("collabFloorRC", xmRowColumnWidgetClass,
                   wgFlrFrame,
                   XmNorientation, XmVERTICAL,
                   XmNisAligned, True,
                   XmNentryAlignment, XmALIGNMENT_CENTER,
                   NULL);
    str = XmStringCreateSimple("Floor");
    wgFlrTitle = XtVaCreateManagedWidget("floorTitle", xmLabelWidgetClass,
                       wgFloorRC,
                       XmNlabelString, str,
                       NULL);
```

```
  XmStringFree(str);
  wgFlrTglRC = togglesCreate(wgFloorRC, "floorTgls", TGL_CHECK,
      XmHORIZONTAL,
                collabFloorTgls, NULL);
  XtOverrideTranslations(collabFloorTgls[0].wgTgl, pXlns);
  XtManageChild(wgFlrTglRC);

  str = XmStringCreateSimple("<Leader>");
  wgCollabFloorLabel = XtVaCreateManagedWidget("floorLabel",
                          xmLabelWidgetClass,
                          wgFloorRC,
                          XmNlabelString, str,
                          NULL);
  XmStringFree(str);
  wgFlrCmdRC = buttonsCreate(wgFloorRC, "floorCmds", XmVERTICAL,
                flrCmdBtns, NULL);
  XtManageChild(wgFlrCmdRC);
  XtManageChild(wgFloorRC);
  XtManageChild(wgFlrFrame);

  XtManageChild(wgIxnForm);

  n=0;
  XtSetArg(args[n], XmNrows, 5);n++;
  XtSetArg(args[n], XmNcolumns, 32);n++;
  XtSetArg(args[n], XmNeditable, False);n++;
  XtSetArg(args[n], XmNeditMode, XmMULTI_LINE_EDIT);n++;
  XtSetArg(args[n], XmNscrollingPolicy, XmAUTOMATIC); n++;
  XtSetArg(args[n], XmNvisualPolicy, XmCONSTANT); n++;
  XtSetArg(args[n], XmNscrollBarDisplayPolicy, XmAS_NEEDED); n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_WIDGET);n++;
  XtSetArg(args[n], XmNtopWidget, wgIxnForm);n++;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNbottomAttachment, XmATTACH_FORM);n++;
  XtSetArg(args[n], XmNscrollHorizontal, False); n++;
  XtSetArg(args[n], XmNscrollVertical, True); n++;
  XtSetArg(args[n], XmNwordWrap, True); n++;

  wgDbgText = createMessageBuffer(wgParent, "collabTextMsgs",
                &collabMsgBufCntl, args,n);
  XtManageChild(wgDbgText);
}

static void
collabSetPermModeCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
  collabSetPermOprn((CollabPermission)xpClient, cbs->set);
}
```

```
static void
collabSetIxnModeCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  collabSetIxnModeOprn((CollabIxnMode)xpClient);
}

static void
collabSetSesFormatCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  collabSetSesFormatOprn((CollabFmtMode)xpClient);
}

static void
collabSetFloorModeCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  collabSetFloorModeOprn((CollabFlrMode)xpClient);
}

static void
collabFreeFloorCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
  collabFreeFloorOprn(cbs->set);
}

static void
collabRequestFloorCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
  collabRequestFloorOprn();
}

static void
collabFloorParamsCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
/*control panel*/
}
```

```
Widget
frontCollabsCB(wgTgl, pMgrCD, xpFoo)
     Widget          wgTgl;
     mgrCntlData *pMgrCD;
     XtPointer xpFoo;
{
  Widget wgShell;
  panelCntlData       *pPanelCntl;
  int fToggles, iColl;
  char **sbNames;

  if (pMgrCD->wgCntl) {
    return;
  }
  pMgrCD->wgTgl = wgTgl;

  pPanelCntl = (panelCntlData *) malloc(sizeof(panelCntlData));
  memset(pPanelCntl, 0, sizeof(panelCntlData));

  pPanelCntl->sName = "Collab";
  pPanelCntl->fnMenuBar = createCollabMenuBar;
  pPanelCntl->fnChooseCB = chooseOneCollabCB;
  pPanelCntl->fCntlArea = True;

  fToggles = PANEL_SELECT | PANEL_RENAME | PANEL_LEAVE | PANEL_TERMINATE |
    PANEL_UNSELECT;
  pMgrCD->wgCntl = wgShell =
    createPanelControl(pMgrCD->wgParent, "collabControl", wgTgl, pPanelCntl
        ,
              fToggles, PANEL_CHOOSEONE, NULL);

  createCollabCntlAreaCB(pPanelCntl->wgCntlArea, NULL, NULL);

  pCollabDismissData = pPanelCntl->pDismiss;
  pcoCollab = pPanelCntl->pChooseOne;

  iColl = getCollabIndex(0);
  if(iColl != -1){
    sbNames = getCollabNameList(0);
    chooseOneChangeList(pcoCollab, sbNames, iColl);
    if (sbNames) {
      free(sbNames);
    }
  }
  wgCollabShell = wgShell;
  return wgShell;
}

static void
collabDismissCB(wg, xpClient, cbs)
     Widget          wg;
```

```
    XtPointer       xpClient;
    XmPushButtonCallbackStruct *cbs;
{
  defaultShellDismissCB(wg, (XtPointer)pCollabDismissData, cbs);
}

static void
collabCmdCB(wg, xpClient, cbs)
    Widget wg;
    XtPointer xpClient;
    XmPushButtonCallbackStruct *cbs;
{
  static multiChooseCntlData inviteCD, initiateCD, permSetCD;
  static optChooseCntlData joinCD, removeCD, connectCD;
  static optChooseCntlData permChkCD, floorSetCD, describeCD;
  CollabCmd iCollabCmd;

  iCollabCmd = (CollabCmd) xpClient;
  switch (iCollabCmd) {
  case CollabCmd_INITIATE:
    initiateCD.fnCallback = collabInitiateCMCB;
    initiateCD.xpClient = (XtPointer) NULL;
    systemGenChooseManyCB(wg, (XtPointer) & initiateCD, NULL);
    break;
  case CollabCmd_INVITE:
    inviteCD.fnCallback = collabInviteCMCB;
    inviteCD.xpClient = (XtPointer) NULL;
    systemGenChooseManyCB(wg, (XtPointer) & inviteCD, NULL);
    break;
  case CollabCmd_JOIN:
    joinCD.fnCallback = collabJoinCOCB;
    joinCD.xpClient = (XtPointer) NULL;
    sesMgrGenChooseOneCB(wg, (XtPointer) & joinCD, NULL);
    break;
  case CollabCmd_REMOVE:
    removeCD.fnCallback = collabRemoveCOCB;
    removeCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseOneCB(wg, (XtPointer) & removeCD, NULL);
    break;
  case CollabCmd_CONNECT:
    connectCD.fnCallback = collabConnectCOCB;
    connectCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseOneCB(wg, (XtPointer) & connectCD, NULL);
    break;
  case CollabCmd_DESCRIBE:
    describeCD.fnCallback = collabDescribeCOCB;
    describeCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseOneCB(wg, (XtPointer) & describeCD, NULL);
    break;
  case CollabCmd_PERMSET:
    permSetCD.fnCallback = collabPermSetCMCB;
    permSetCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseManyCB(wg, (XtPointer) & permSetCD, NULL);
```

```
    break;
  case CollabCmd_PERMCHK:
    permChkCD.fnCallback = collabPermCheckCOCB;
    permChkCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseOneCB(wg, (XtPointer) & permChkCD, NULL);
    break;
  case CollabCmd_FLOOR:
    floorSetCD.fnCallback = collabFloorSetCOCB;
    floorSetCD.xpClient = (XtPointer) NULL;
    collabFrontGenChooseOneCB(wg, (XtPointer) & floorSetCD, NULL);
    break;
  }
}

static void
collabInitiateCMCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             *aiWhich = (int*) xpCall;

  collabInitiateOprn(aiWhich);
}

static void
collabInviteCMCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             *aiWhich = (int*) xpCall;

  collabInviteOprn(aiWhich);
}

static void
collabJoinCOCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabAskJoinOprn(iWhich);
}

static void
collabRemoveCOCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabRemoveOprn(iWhich);
}
```

```
static void
collabConnectCOCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabCommConnectOprn(iWhich);
}

static void
collabPermSetCMCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             *aiWhich = (int*) xpCall;

  collabSetPermsDialogPopup(wg, (XtPointer)aiWhich, NULL);
}

static void
collabSetPermsCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             *aiWhich = (int*) xpClient;

  collabPermSetOprn(aiWhich);
}

static void
collabPermCheckCOCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabPermCheckOprn(iWhich);
}

static void
collabDescribeCOCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabDescribeOprn(iWhich);
}

static void
collabFloorSetCOCB(wg, xpClient, xpCall)
```

```
     Widget          wg;
     XtPointer       xpClient, xpCall;
{
  int             iWhich = (int) xpCall;

  collabFloorSetOprn(iWhich);
}

static void
leaveCollabCB(wg, xpClient, cbs)
     Widget wg;
     XtPointer xpClient;
     XmAnyCallbackStruct *cbs;
{
  panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;

  collabLeaveOprn((int)pGenCD->xpCall);
}

static void
terminateCollabCB(wg, xpClient, cbs)
     Widget wg;
     XtPointer xpClient;
     XmAnyCallbackStruct *cbs;
{
  panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;

  collabTerminateOprn((int)pGenCD->xpCall);
}

static void
renameCollabCB(wg, xpClient, cbs)
     Widget wg;
     XtPointer xpClient;
     XmAnyCallbackStruct *cbs;
{
  panelAxnCntlData *pGenCD = (panelAxnCntlData*)xpClient;

  collabRenameOprn((int)pGenCD->xpCall, 0);
}

static void
chooseOneCollabCB(wg, xpClientData, xpCallData)
     Widget          wg;
     XtPointer       xpClientData, xpCallData;
{
  int            which = (int) xpCallData;
  panelCntlData       *pPanelCntl = (panelCntlData *) xpClientData;
  static panelAxnCntlData genCDConfirm;
  static panelAxnCntlData genCDRen;

  switch (pPanelCntl->iMode) {
  case PANEL_SELECT:
```

```
    collabSelectOprn(which);
    break;
  case PANEL_UNSELECT:
    collabUnselectOprn(which);
    break;
  case PANEL_RENAME:
    genCDRen.fnCallback = renameCollabCB;
    genCDRen.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDRen;
    panelDefaultRenamePUCB(wg, xpClientData, xpCallData);
    break;
  case PANEL_LEAVE:
    genCDConfirm.fnCallback = leaveCollabCB;
    genCDConfirm.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDConfirm;
    panelDefaultConfirmPUCB(wg, xpClientData, xpCallData);
    break;
  case PANEL_TERMINATE:
    genCDConfirm.fnCallback = terminateCollabCB;
    genCDConfirm.xpCall = xpCallData;
    pPanelCntl->xpClient = (XtPointer)&genCDConfirm;
    panelDefaultConfirmPUCB(wg, xpClientData, xpCallData);
    break;
  default:
    break;
  }
}

static void
collabCreateBrowserCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  /*popup a browser dialog per session*/
}

static void
collabOperationCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
  mgrCntlData *mgrCntl = (mgrCntlData*)xpClient;
  collabOperationsOprn(mgrCntl, cbs->set);
}

static void
collabOptionCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
```

```
  static Widget wgCurrInvite, wgCurrJoin;
  CollabOption iCollOpt = (CollabOption)xpClient;

  switch(iCollOpt){
  case CollabOpt_ACCEPT:
  case CollabOpt_DECLINE:
    if(cbs->set){
      if(wgCurrInvite && (wgCurrInvite != wg)){
    XmToggleButtonSetState(wgCurrInvite, False, True);
      }
      wgCurrInvite = wg;
      collabSetInviteOption(iCollOpt);
    }
    else if(cbs->event){
      collabSetInviteOption(CollabOpt_Null);
    }
    break;
  case CollabOpt_ALLOW:
  case CollabOpt_DENY:
    if(cbs->set){
      if(wgCurrJoin && (wgCurrJoin != wg)){
    XmToggleButtonSetState(wgCurrJoin, False, True);
      }
      collabSetAskJoinOption(iCollOpt);
      wgCurrJoin = wg;
    }
    else if(cbs->event){
      collabSetAskJoinOption(CollabOpt_Null);
    }
    break;
  case CollabOpt_START:
    if(cbs->set){
      collabSetStartOption(iCollOpt);
    }
    else{
      collabSetStartOption(CollabOpt_Null);
    }
    break;
  case CollabOpt_FORCEJOIN:
    if(cbs->set){
      collabSetForceJoinOption(iCollOpt);
    }
    else{
      collabSetForceJoinOption(CollabOpt_Null);
    }
    break;
  }
}

static void
collabOptJoinCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
```

```
     XmToggleButtonCallbackStruct *cbs;
{
  static Widget wgCurrSet;

  if (cbs->set) {
    if(wgCurrSet && (wgCurrSet != wg)){
      XmToggleButtonSetState(wgCurrSet, False, True);
    }
    wgCurrSet = wg;
  }
}

static void
collabShowTraceCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmToggleButtonCallbackStruct *cbs;
{
  fDebugTrace = cbs->set;
}

static void
collabSyncCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  collabSyncOprn((CollabCmd)xpClient);
}


static Widget
createCollCommMenuBar(wgParent, sName, pCommCD, argList)
     Widget wgParent;
     char *sName;
     shaCommCntlData *pCommCD;
     XtVarArgsList   argList;
{
  Widget wgMenuBar, wgControlPD, wgOptionsPD, wgHelpPD;
  static menuItem controlPD[] = {
    {"Clear", (XtPointer) NULL, False, collabCommClearCB},
    {"Dismiss", (XtPointer) NULL, False, collabCommDismissCB},
    {"Close", (XtPointer) NULL, False, collabCommTerminateCB},
    {NULL}
  };
  static menuItem optionsPD[] = {
    {"Comm. Panel", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  menuItem *pControlPD, *pOptionsPD;
  Arg             args[8];
  int             n;
```

```
  n = 0;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;
  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);

  pControlPD = pCommCD->pControlPD = (menuItem*)malloc(sizeof(controlPD));
  memcpy(pControlPD, controlPD, sizeof(controlPD));
  pControlPD[0].xpClient = (XtPointer)pCommCD;
  pControlPD[1].xpClient = (XtPointer)pCommCD;
  pControlPD[2].xpClient = (XtPointer)pCommCD;
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                  pCommCD->pControlPD, NULL);
  pOptionsPD = pCommCD->pOptionsPD = (menuItem*)malloc(sizeof(optionsPD));
  memcpy(pOptionsPD, optionsPD, sizeof(optionsPD));
  pOptionsPD[0].xpClient = (XtPointer)pCommCD;
  pOptionsPD[1].xpClient = (XtPointer)pCommCD;
  wgOptionsPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                  pCommCD->pOptionsPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}

static void
collabCommTerminateCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCleanUpCB(wg, pCommCD->pDialogCD, cbs);
  XtDestroyWidget(pCommCD->pDialogCD->wgDialog);
  XtFree((char*)pCommCD->pDialogCD);
  if(pCommCD->pOptionsPD){
    XtFree((char*)pCommCD->pOptionsPD);
        pCommCD->pOptionsPD = NULL;
  }
  if(pCommCD->pControlPD){
    XtFree((char*)pCommCD->pControlPD);
        pCommCD->pControlPD = NULL;
  }
    if(pCommCD->pRemTextCD != NULL){
      XtFree((char*)pCommCD->pRemTextCD);
        pCommCD->pRemTextCD = NULL;
    }
    if(pCommCD->pLocTextCD != NULL){
      XtFree((char*)pCommCD->pLocTextCD);
        pCommCD->pLocTextCD = NULL;
    }
```

```
  collabCommDisconnectOprn(pCommCD->smSIdTag, pCommCD->remSIdTag);
}

static void
collabCommDismissCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  defaultDialogCancelCB(wg, (XtPointer)pCommCD->pDialogCD, cbs);
}

static void
collabCommClearCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  clearMessageBuffer(pCommCD->pLocTextCD);
  collabCommSendMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, "");
}

static void
collabCommTextCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = XmTextGetString(wg);
  collabCommSendMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, str);
  XtFree(str);
}


void
setupCollabCommDialog(pCommCD)
     shaCommCntlData *pCommCD;
{
  static buttonItem dlgBtns[] = {
    {"Dismiss", (XtPointer) DIALOG_OK, collabCommDismissCB},
    {"Close", (XtPointer) DIALOG_CANCEL, collabCommTerminateCB},
    {"Help", (XtPointer) DIALOG_HELP, NULL},
    {NULL}
  };

  pCommCD->fnMenuBar = createCollCommMenuBar;
  pCommCD->fnTextCallback = collabCommTextCB;
```

```
  pCommCD->pDlgBtns = dlgBtns;
  createShastraCommDialog(pCommCD);
}


static Widget
createCollInviteCommMenuBar(wgParent, sName, pCommCD, argList)
     Widget wgParent;
     char *sName;
     shaCommCntlData *pCommCD;
     XtVarArgsList   argList;
{
  Widget wgMenuBar, wgControlPD, wgOptionsPD, wgHelpPD;
  static menuItem controlPD[] = {
    {"Clear", (XtPointer) NULL, False, collabInviteCommClearCB},
    {"Dismiss", (XtPointer) NULL, False, collabInviteCommDismissCB},
    {"Close", (XtPointer) NULL, False, collabInviteCommTerminateCB},
    {NULL}
  };
  static menuItem optionsPD[] = {
    {"Comm. Panel", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  menuItem *pControlPD, *pOptionsPD;
  Arg             args[8];
  int             n;

  n = 0;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;
  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);

  pControlPD = pCommCD->pControlPD = (menuItem*)malloc(sizeof(controlPD));
  memcpy(pControlPD, controlPD, sizeof(controlPD));
  pControlPD[0].xpClient = (XtPointer)pCommCD;
  pControlPD[1].xpClient = (XtPointer)pCommCD;
  pControlPD[2].xpClient = (XtPointer)pCommCD;
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                   pCommCD->pControlPD, NULL);
  pOptionsPD = pCommCD->pOptionsPD = (menuItem*)malloc(sizeof(optionsPD));
  memcpy(pOptionsPD, optionsPD, sizeof(optionsPD));
  pOptionsPD[0].xpClient = (XtPointer)pCommCD;
  pOptionsPD[1].xpClient = (XtPointer)pCommCD;
  wgOptionsPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                   pCommCD->pOptionsPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}
```

```
static void
collabInviteCommDismissCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  defaultDialogCancelCB(wg, (XtPointer)pCommCD->pDialogCD, cbs);
}


static void
collabInviteCommTerminateCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCleanUpCB(wg, pCommCD->pDialogCD, cbs);
  XtDestroyWidget(pCommCD->pDialogCD->wgDialog);
  XtFree((char*)pCommCD->pDialogCD);
  if(pCommCD->pOptionsPD){
    XtFree((char*)pCommCD->pOptionsPD);
        pCommCD->pOptionsPD = NULL;
  }
  if(pCommCD->pControlPD){
    XtFree((char*)pCommCD->pControlPD);
        pCommCD->pControlPD = NULL;
  }
    if(pCommCD->pRemTextCD != NULL){
      XtFree((char*)pCommCD->pRemTextCD);
        pCommCD->pRemTextCD = NULL;
    }
    if(pCommCD->pLocTextCD != NULL){
      XtFree((char*)pCommCD->pLocTextCD);
        pCommCD->pLocTextCD = NULL;
    }
  collabDeleteInvitePanelOprn(pCommCD->smSIdTag);
}

static void
collabInviteCommClearCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = "";
```

```
  clearMessageBuffer(pCommCD->pLocTextCD);
  if(pCommCD->pSIdTags){
    collabSendInviteMessageOprn(pCommCD->smSIdTag, pCommCD->pSIdTags,
                str);
  }
  else{
    collabSendUniInviteMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag,
                   str);
  }
}

static void
collabInviteCommTextCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = XmTextGetString(wg);
  if(pCommCD->pSIdTags){
    collabSendInviteMessageOprn(pCommCD->smSIdTag, pCommCD->pSIdTags,
                str);
  }
  else{
    collabSendUniInviteMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag,
                   str);
  }
  XtFree(str);
}

void
setupCollabInviteCommDialog(pCommCD)
     shaCommCntlData *pCommCD;
{
  static buttonItem dlgBtns[] = {
    {"Dismiss", (XtPointer) DIALOG_OK, collabInviteCommDismissCB},
    {"Close", (XtPointer) DIALOG_CANCEL, collabInviteCommTerminateCB},
    {"Help", (XtPointer) DIALOG_HELP, NULL},
    {NULL}
  };

  pCommCD->fnMenuBar = createCollInviteCommMenuBar;
  pCommCD->fnTextCallback = collabInviteCommTextCB;

  pCommCD->pDlgBtns = dlgBtns;
  createShastraCommDialog(pCommCD);
}


static Widget
createCollInvRespCommMenuBar(wgParent, sName, pCommCD, argList)
```

```
     Widget wgParent;
     char *sName;
     shaCommCntlData *pCommCD;
     XtVarArgsList   argList;
{
  Widget wgMenuBar, wgControlPD, wgOptionsPD, wgHelpPD;
  static menuItem controlPD[] = {
    {"Clear", (XtPointer) NULL, False, collabInvRespCommClearCB},
    {"Accept", (XtPointer) NULL, False, collabInvRespCommAcceptCB},
    {"Decline", (XtPointer) NULL, False, collabInvRespCommDeclineCB},
    {NULL}
  };
  static menuItem optionsPD[] = {
    {"Comm. Panel", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  menuItem *pControlPD, *pOptionsPD;
  Arg             args[8];
  int             n;

  n = 0;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;
  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);

  pControlPD = pCommCD->pControlPD = (menuItem*)malloc(sizeof(controlPD));
  memcpy(pControlPD, controlPD, sizeof(controlPD));
  pControlPD[0].xpClient = (XtPointer)pCommCD;
  pControlPD[1].xpClient = (XtPointer)pCommCD;
  pControlPD[2].xpClient = (XtPointer)pCommCD;
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                   pCommCD->pControlPD, NULL);
  pOptionsPD = pCommCD->pOptionsPD = (menuItem*)malloc(sizeof(optionsPD));
  memcpy(pOptionsPD, optionsPD, sizeof(optionsPD));
  pOptionsPD[0].xpClient = (XtPointer)pCommCD;
  pOptionsPD[1].xpClient = (XtPointer)pCommCD;
  wgOptionsPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                   pCommCD->pOptionsPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}


static void
collabInvRespCommTerminateCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
```

```
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCleanUpCB(wg, pCommCD->pDialogCD, cbs);
  XtDestroyWidget(pCommCD->pDialogCD->wgDialog);
  XtFree((char*)pCommCD->pDialogCD);
  if(pCommCD->pOptionsPD){
    XtFree((char*)pCommCD->pOptionsPD);
        pCommCD->pOptionsPD = NULL;
  }
  if(pCommCD->pControlPD){
    XtFree((char*)pCommCD->pControlPD);
        pCommCD->pControlPD = NULL;
  }
    if(pCommCD->pRemTextCD != NULL){
      XtFree((char*)pCommCD->pRemTextCD);
        pCommCD->pRemTextCD = NULL;
    }
    if(pCommCD->pLocTextCD != NULL){
      XtFree((char*)pCommCD->pLocTextCD);
        pCommCD->pLocTextCD = NULL;
    }
}


static void
collabInvRespCommAcceptCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  char *str;
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  str = "Invitation Accepted\n";
/*
  collabSendInvRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, str);
  */
  collabInvRespCommTerminateCB(wg, xpClient, cbs);
  collabInviteAcceptOprn(pCommCD->smSIdTag, pCommCD->remSIdTag);
}


static void
collabInvRespCommDeclineCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  char *str;
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  str = "Invitation Declined\n";
/*
```

```
  collabSendInvRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, str);
  */
  collabInvRespCommTerminateCB(wg, xpClient, cbs);
  collabInviteDeclineOprn(pCommCD->smSIdTag, pCommCD->remSIdTag);
}

static void
collabInvRespCommClearCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  clearMessageBuffer(pCommCD->pLocTextCD);
  collabSendInvRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, "");
}

static void
collabInvRespCommTextCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = XmTextGetString(wg);
  collabSendInvRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, str);
  XtFree(str);
}

void
setupCollabInvRespCommDialog(pCommCD)
     shaCommCntlData *pCommCD;
{
  static buttonItem dlgBtns[] = {
    {"Accept", (XtPointer) DIALOG_OK, collabInvRespCommAcceptCB},
    {"Decline", (XtPointer) DIALOG_CANCEL, collabInvRespCommDeclineCB},
    {"Help", (XtPointer) DIALOG_HELP, NULL},
    {NULL}
  };

  pCommCD->fnMenuBar = createCollInvRespCommMenuBar;
  pCommCD->fnTextCallback = collabInvRespCommTextCB;
  pCommCD->fnCancelCallback = collabInvRespCommDeclineCB;
/*
  pCommCD->fBehave = DIALOG_AUTOLOWER;
  pCommCD->iDelay = 60000;
  */

  pCommCD->pDlgBtns = dlgBtns;
  createShastraCommDialog(pCommCD);
```

```
}



static Widget
createCollAskJnRespCommMenuBar(wgParent, sName, pCommCD, argList)
     Widget wgParent;
     char *sName;
     shaCommCntlData *pCommCD;
     XtVarArgsList   argList;
{
  Widget wgMenuBar, wgControlPD, wgOptionsPD, wgHelpPD;
  static menuItem controlPD[] = {
    {"Clear", (XtPointer) NULL, False, collabAskJnRespCommClearCB},
    {"Allow", (XtPointer) NULL, False, collabAskJnRespCommAllowCB},
    {"Deny", (XtPointer) NULL, False, collabAskJnRespCommDenyCB},
    {NULL}
  };
  static menuItem optionsPD[] = {
    {"Comm. Panel", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  menuItem *pControlPD, *pOptionsPD;
  Arg             args[8];
  int             n;

  n = 0;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;
  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);

  pControlPD = pCommCD->pControlPD = (menuItem*)malloc(sizeof(controlPD));
  memcpy(pControlPD, controlPD, sizeof(controlPD));
  pControlPD[0].xpClient = (XtPointer)pCommCD;
  pControlPD[1].xpClient = (XtPointer)pCommCD;
  pControlPD[2].xpClient = (XtPointer)pCommCD;
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                  pCommCD->pControlPD, NULL);
  pOptionsPD = pCommCD->pOptionsPD = (menuItem*)malloc(sizeof(optionsPD));
  memcpy(pOptionsPD, optionsPD, sizeof(optionsPD));
  pOptionsPD[0].xpClient = (XtPointer)pCommCD;
  pOptionsPD[1].xpClient = (XtPointer)pCommCD;
  wgOptionsPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                  pCommCD->pOptionsPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}

static void
```

```
collabAskJnRespCommTerminateCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCleanUpCB(wg, pCommCD->pDialogCD, cbs);
  XtDestroyWidget(pCommCD->pDialogCD->wgDialog);
  XtFree((char*)pCommCD->pDialogCD);
  if(pCommCD->pOptionsPD){
    XtFree((char*)pCommCD->pOptionsPD);
        pCommCD->pOptionsPD = NULL;
  }
  if(pCommCD->pControlPD){
    XtFree((char*)pCommCD->pControlPD);
        pCommCD->pControlPD = NULL;
  }
    if(pCommCD->pRemTextCD != NULL){
      XtFree((char*)pCommCD->pRemTextCD);
        pCommCD->pRemTextCD = NULL;
    }
    if(pCommCD->pLocTextCD != NULL){
      XtFree((char*)pCommCD->pLocTextCD);
        pCommCD->pLocTextCD = NULL;
    }
}


static void
collabAskJnRespCommAllowCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  char *str;
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  str = "Requested Join Allowed\n";
/*
  collabSendAskJnRespMessageOprn(pCommCD->smSIdTag,
  pCommCD->remSIdTag, str);
  */
  collabAskJnRespCommTerminateCB(wg, xpClient, cbs);
  collabAskJoinAllowOprn(pCommCD->smSIdTag, pCommCD->remSIdTag);
}

static void
collabAskJnRespCommDenyCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
```

```
  char *str;
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  str = "Requested Join Denied\n";
/*
  collabSendAskJnRespMessageOprn(pCommCD->smSIdTag,
  pCommCD->remSIdTag, str);
  */
  collabAskJnRespCommTerminateCB(wg, xpClient, cbs);
  collabAskJoinDenyOprn(pCommCD->smSIdTag, pCommCD->remSIdTag);
}

static void
collabAskJnRespCommClearCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  clearMessageBuffer(pCommCD->pLocTextCD);
  collabSendAskJnRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, "")
      ;
}

static void
collabAskJnRespCommTextCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = XmTextGetString(wg);
  collabSendAskJnRespMessageOprn(pCommCD->smSIdTag, pCommCD->remSIdTag, str
      );
  XtFree(str);
}

void
setupCollabAskJnRespCommDialog(pCommCD)
     shaCommCntlData *pCommCD;
{
  static buttonItem dlgBtns[] = {
    {"Allow", (XtPointer) DIALOG_OK, collabAskJnRespCommAllowCB},
    {"Deny", (XtPointer) DIALOG_CANCEL, collabAskJnRespCommDenyCB},
    {"Help", (XtPointer) DIALOG_HELP, NULL},
    {NULL}
  };

  pCommCD->fnMenuBar = createCollAskJnRespCommMenuBar;
  pCommCD->fnTextCallback = collabAskJnRespCommTextCB;
```

```
  pCommCD->fnCancelCallback = collabAskJnRespCommDenyCB;
/*
  pCommCD->fBehave = DIALOG_AUTOLOWER;
  pCommCD->iDelay = 60000;
  */

  pCommCD->pDlgBtns = dlgBtns;
  createShastraCommDialog(pCommCD);
}


static Widget
createCollAskJoinCommMenuBar(wgParent, sName, pCommCD, argList)
     Widget wgParent;
     char *sName;
     shaCommCntlData *pCommCD;
     XtVarArgsList   argList;
{
  Widget wgMenuBar, wgControlPD, wgOptionsPD, wgHelpPD;
  static menuItem controlPD[] = {
    {"Clear", (XtPointer) NULL, False, collabAskJoinCommClearCB},
    {"Dismiss", (XtPointer) NULL, False, collabAskJoinCommDismissCB},
    {"Close", (XtPointer) NULL, False, collabAskJoinCommTerminateCB},
    {NULL}
  };
  static menuItem optionsPD[] = {
    {"Comm. Panel", (XtPointer) NULL, False, NULL},
    {NULL}
  };
  menuItem *pControlPD, *pOptionsPD;
  Arg             args[8];
  int             n;

  n = 0;
  XtSetArg(args[n], XmNrightAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNleftAttachment, XmATTACH_FORM);
  n++;
  XtSetArg(args[n], XmNtopAttachment, XmATTACH_FORM);
  n++;
  wgMenuBar = XmCreateMenuBar(wgParent, sName, args, n);

  pControlPD = pCommCD->pControlPD = (menuItem*)malloc(sizeof(controlPD));
  memcpy(pControlPD, controlPD, sizeof(controlPD));
  pControlPD[0].xpClient = (XtPointer)pCommCD;
  pControlPD[1].xpClient = (XtPointer)pCommCD;
  pControlPD[2].xpClient = (XtPointer)pCommCD;
  wgControlPD = pulldownMenuCreate(wgMenuBar, "Control", MENU_PUSH,
                   pCommCD->pControlPD, NULL);
  pOptionsPD = pCommCD->pOptionsPD = (menuItem*)malloc(sizeof(optionsPD));
  memcpy(pOptionsPD, optionsPD, sizeof(optionsPD));
  pOptionsPD[0].xpClient = (XtPointer)pCommCD;
  pOptionsPD[1].xpClient = (XtPointer)pCommCD;
```

```
  wgOptionsPD = pulldownMenuCreate(wgMenuBar, "Options", MENU_CHECK,
                    pCommCD->pOptionsPD, NULL);
  wgHelpPD = createHelpPD(wgMenuBar);
  return wgMenuBar;
}


static void
collabAskJoinCommDismissCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCancelCB(wg, (XtPointer)pCommCD->pDialogCD, cbs);
}


static void
collabAskJoinCommTerminateCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmPushButtonCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  defaultDialogCleanUpCB(wg, pCommCD->pDialogCD, cbs);
  XtDestroyWidget(pCommCD->pDialogCD->wgDialog);
  XtFree((char*)pCommCD->pDialogCD);
  if(pCommCD->pOptionsPD){
    XtFree((char*)pCommCD->pOptionsPD);
        pCommCD->pOptionsPD = NULL;
  }
  if(pCommCD->pControlPD){
    XtFree((char*)pCommCD->pControlPD);
        pCommCD->pControlPD = NULL;
  }
    if(pCommCD->pRemTextCD != NULL){
      XtFree((char*)pCommCD->pRemTextCD);
        pCommCD->pRemTextCD = NULL;
    }
    if(pCommCD->pLocTextCD != NULL){
      XtFree((char*)pCommCD->pLocTextCD);
        pCommCD->pLocTextCD = NULL;
    }
  collabDeleteAskJoinPanelOprn(pCommCD->smSIdTag);
}

static void
collabAskJoinCommClearCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
```

```
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;

  clearMessageBuffer(pCommCD->pLocTextCD);
  collabSendAskJoinMessageOprn(pCommCD->smSIdTag, "");
}

static void
collabAskJoinCommTextCB(wg, xpClient, cbs)
     Widget          wg;
     XtPointer       xpClient;
     XmAnyCallbackStruct *cbs;
{
  shaCommCntlData *pCommCD = (shaCommCntlData*)xpClient;
  char *str;

  str = XmTextGetString(wg);
  collabSendAskJoinMessageOprn(pCommCD->smSIdTag, str);
  XtFree(str);
}

void
setupCollabAskJoinCommDialog(pCommCD)
     shaCommCntlData *pCommCD;
{
  static buttonItem dlgBtns[] = {
    {"Dismiss", (XtPointer) DIALOG_OK, collabAskJoinCommDismissCB},
    {"Close", (XtPointer) DIALOG_CANCEL, collabAskJoinCommTerminateCB},
    {"Help", (XtPointer) DIALOG_HELP, NULL},
    {NULL}
  };

  pCommCD->fnMenuBar = createCollAskJoinCommMenuBar;
  pCommCD->fnTextCallback = collabAskJoinCommTextCB;

  pCommCD->pDlgBtns = dlgBtns;
  createShastraCommDialog(pCommCD);
}



/*
 * Function --
 */
void
showCollabInfo(s)
     char *s;
{
  if(collabMsgBufCntl.wgText && fDebugTrace){
    wprintf(&collabMsgBufCntl,"%s", s);
  }
}
```

```
void
collabUtilPopupMessage(msg)
     char *msg;
{
  static dialogCntlData infoDlgCD;

  showCollabInfo(msg);
  if(infoDlgCD.wgDialog == NULL){
    infoDlgCD.fFlags = DIALOG_OK;
    infoDlgCD.fBehave = DIALOG_AUTOLOWER;
    infoDlgCD.iDelay = 5000;
    infoDlgCD.sName = "Session Information";
    infoDlgCD.sMessage = "Yo, User Dude!\nThis is, like, cool!!";

    createInformationDialog(pFrontAppData->wgTop, "infoDialog",
                &infoDlgCD, NULL);
  }
  defaultDialogPopupMessage(&infoDlgCD, msg);
}

static void
collabSetPermsDialogPopup(wg, xpClient, xpCall)
     Widget wg;
     XtPointer xpClient, xpCall;
{
  static dialogCntlData permDlgCD;
  Widget wgPermTglRC;
  static toggleItem permTgls[] = {
    {"Access",(XtPointer)CollabPerm_ACCESS, True, collabSetPermModeCB},
    {"Browse",(XtPointer)CollabPerm_BROWSE, True, collabSetPermModeCB},
    {"Modify",(XtPointer)CollabPerm_MODIFY, True, collabSetPermModeCB},
    {"Copy",(XtPointer)CollabPerm_COPY, False, collabSetPermModeCB},
    {"Grant",(XtPointer)CollabPerm_GRANT, False, collabSetPermModeCB},
    {NULL}
  };

  if(permDlgCD.wgDialog == NULL){
    permDlgCD.fFlags = DIALOG_OK | DIALOG_CANCEL | DIALOG_HELP;
    permDlgCD.fMode = XmDIALOG_APPLICATION_MODAL;
    permDlgCD.sName = "Session Permissions";
    permDlgCD.sMessage = "Set Session Permissions";
    permDlgCD.xpClient = xpClient;
    permDlgCD.fnCallback = collabSetPermsCB;

    createTemplateDialog(wgCollabShell, "permDialog", &permDlgCD, NULL);

    wgPermTglRC = togglesCreate(permDlgCD.wgDialog, "permTgls",
                TGL_CHECK, XmHORIZONTAL, permTgls, NULL);
    XtManageChild(wgPermTglRC);
  }
  defaultDialogPopup(&permDlgCD);
}
```

```
void
collabCheckPermsDialogPopup(wg, xpClient, xpCall)
     Widget wg;
     XtPointer xpClient, xpCall;
{
  static dialogCntlData permDlgCD;
  Widget wgPermTglRC;
  char *str, msgBuf[128];
  static toggleItem permTgls[] = {
    {"Access",(XtPointer)CollabPerm_ACCESS, False, NULL},
    {"Browse",(XtPointer)CollabPerm_BROWSE, False, NULL},
    {"Modify",(XtPointer)CollabPerm_MODIFY, False, NULL},
    {"Copy",(XtPointer)CollabPerm_COPY, False, NULL},
    {"Grant",(XtPointer)CollabPerm_GRANT, False, NULL},
    {NULL}
  };

  if(permDlgCD.wgDialog == NULL){
    permDlgCD.fFlags = DIALOG_OK | DIALOG_HELP;
    permDlgCD.sName = "Session Permissions";
    permDlgCD.sMessage = "Session Permissions";
    permDlgCD.xpClient = xpClient;
    permDlgCD.fnCallback = collabSetPermsCB;
    permDlgCD.fBehave = DIALOG_AUTOLOWER;
    permDlgCD.iDelay = 10000;

    createTemplateDialog(wgCollabShell, "permDialog", &permDlgCD, NULL);

    wgPermTglRC = togglesCreate(permDlgCD.wgDialog, "permTgls",
                TGL_CHECK, XmHORIZONTAL, permTgls, NULL);
    XtManageChild(wgPermTglRC);
  }
  /*set tgl values from flags, name from tag*/
  str = "who?";
  sprintf(msgBuf, "Session Permissions for - %s", str);
  defaultDialogPopupMessage(&permDlgCD, msgBuf);
}



static void
collabGenChooseOneSetup(wg, pOptCD)
     Widget          wg;
     optChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static optChooseCntlData *pChooseOneCD;

  pChooseOneCD = pOptCD;

  if (pcoCollab == NULL) {
    pcoCollab = chooseOneCreate(asDef, coNoInitialHighlight,
```

```
                wg, genCntlChooseCOCB, (XtPointer) & pChooseOneCD, wg,
                "Choose Collab", 200, NULL);
  }
}

static void
collabGenChooseOneCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  collabGenChooseOneSetup(wg, (optChooseCntlData *) xpClient);
  sbNames = getCollabNameList(0);
  chooseOneChangeList(pcoCollab, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseOneMobExec(pcoCollab, wg);
}

static void
systemGenChooseOneSetup(wg, pOptCD)
     Widget          wg;
     optChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static optChooseCntlData *pChooseOneCD;

  pChooseOneCD = pOptCD;

  if (pcoSystems == NULL) {
    pcoSystems = chooseOneCreate(asDef, coNoInitialHighlight,
                 wg, genCntlChooseCOCB, (XtPointer) & pChooseOneCD, wg,
                 "Choose System", 200, NULL);
  }
}

static void
systemGenChooseOneCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  systemGenChooseOneSetup(wg, (optChooseCntlData *) xpClient);
  sbNames = getSystemNameList();
  chooseOneChangeList(pcoSystems, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
```

```
  chooseOneMobExec(pcoSystems, wg);
}

static void
systemGenChooseManySetup(wg, pOptCD)
     Widget          wg;
     multiChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static multiChooseCntlData *pChooseManyCD;

  pChooseManyCD = pOptCD;

  if (pcmSystems == NULL) {
    pcmSystems = chooseManyCreate(asDef, cmNoInitialHighlight,
                 wg, genCntlChooseCOCB, (XtPointer) & pChooseManyCD, wg,
                 "Choose Systems", 200);
  }
}

static void
systemGenChooseManyCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  systemGenChooseManySetup(wg, (multiChooseCntlData *) xpClient);
  sbNames = getSystemNameList();
  chooseManyChangeList(pcmSystems, sbNames, cmNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseManyMobExec(pcmSystems, wg);
}

static void
sesMgrGenChooseOneSetup(wg, pOptCD)
     Widget          wg;
     optChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static optChooseCntlData *pChooseOneCD;

  pChooseOneCD = pOptCD;

  if (pcoSesMgrs == NULL) {
    pcoSesMgrs = chooseOneCreate(asDef, coNoInitialHighlight,
                 wg, genCntlChooseCOCB, (XtPointer) & pChooseOneCD, wg,
                 "Choose SesMgr", 200, NULL);
  }
}
```

```
static void
sesMgrGenChooseOneCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  sesMgrGenChooseOneSetup(wg, (optChooseCntlData *) xpClient);
  sbNames = getSesMgrNameList();
  chooseOneChangeList(pcoSesMgrs, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseOneMobExec(pcoSesMgrs, wg);
}


static void
collabFrontGenChooseOneSetup(wg, pOptCD)
     Widget          wg;
     optChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static optChooseCntlData *pChooseOneCD;

  pChooseOneCD = pOptCD;

  if (pcoCollabFronts == NULL) {
    pcoCollabFronts = chooseOneCreate(asDef, coNoInitialHighlight,
                      wg, genCntlChooseCOCB, (XtPointer) & pChooseOneCD, wg
                          ,
                      "Choose Collab Front", 200, NULL);
  }
}

static void
collabFrontGenChooseOneCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  collabFrontGenChooseOneSetup(wg, (optChooseCntlData *) xpClient);
  sbNames = getCollabFrontNameList(0);
  chooseOneChangeList(pcoCollabFronts, sbNames, coNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseOneMobExec(pcoCollabFronts, wg);
}
```

```
static void
collabFrontGenChooseManySetup(wg, pOptCD)
     Widget          wg;
     multiChooseCntlData *pOptCD;
{
  static String   asDef[] = {NULL};
  static multiChooseCntlData *pChooseManyCD;

  pChooseManyCD = pOptCD;

  if (pcmCollabFronts == NULL) {
    pcmCollabFronts = chooseManyCreate(asDef, cmNoInitialHighlight,
                      wg, genCntlChooseCOCB, (XtPointer) & pChooseManyCD,
                          wg,
                      "Choose Systems", 200);
  }
}

static void
collabFrontGenChooseManyCB(wg, xpClient, xpCall)
     Widget          wg;
     XtPointer       xpClient;
     XtPointer       xpCall;
{
  char          **sbNames;

  collabFrontGenChooseManySetup(wg, (multiChooseCntlData *) xpClient);
  sbNames = getCollabFrontNameList(0);
  chooseManyChangeList(pcmCollabFronts, sbNames, cmNoInitialHighlight);
  if (sbNames) {
    strListDestroy(sbNames);
  }
  chooseManyMobExec(pcmCollabFronts, wg);
}

static void
collabUtilPopupConfirm(wg, pGenCD, sMsg)
     Widget          wg;
     genCntlData    *pGenCD;
     char           *sMsg;
{
  static dialogCntlData dialogCD;
  XtVarArgsList   argList;
  XmString        tmp1, tmp2;

  if (dialogCD.wgDialog == NULL) {

    dialogCD.fFlags = DIALOG_OK | DIALOG_CANCEL;
    dialogCD.fMode = XmDIALOG_APPLICATION_MODAL;
    dialogCD.sMessage = sMsg;
    dialogCD.sName = "Confirm";
    dialogCD.xpClient = pGenCD->xpClient;
```

```
    dialogCD.fnCallback = pGenCD->fnCallback;
    tmp1 = XmStringCreateSimple("Yes");
    tmp2 = XmStringCreateSimple("No");
    argList = XtVaCreateArgsList(NULL,
                XmNokLabelString, tmp1,
                XmNcancelLabelString, tmp2,
                XmNdefaultButtonType, XmDIALOG_CANCEL_BUTTON,
                NULL);
    createQuestionDialog(wg, "confirmDialog", &dialogCD, argList);
    XtFree(argList);
    XmStringFree(tmp1);
    XmStringFree(tmp2);
  }
  defaultDialogPopupMessage(&dialogCD, sMsg);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/errno.h>
#include <netdb.h>
#include <malloc.h>
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>

#ifdef SHASTRA4SUN5
#include <sys/systeminfo.h>
#endif

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

#include <shastra/shastra.h>
#include <shastra/shastraStateDefs.h>

#include <shastra/datacomm/shastraDataH.h>
#include <shastra/datacomm/shastraIdH.h>

#include <shastra/network/hostMgr.h>
#include <shastra/network/server.h>
#include <shastra/network/mplex.h>

#include <shastra/shautils/shautils.h>
```

```
#include <shastra/shautils/kernelFronts.h>

#include <shastra/kernel/kernel_server.h>

#include <shastra/front/frontP.h>
#include <shastra/front/front.h>
#include <shastra/front/front_clientP.h>
#include <shastra/front/front_client.h>
#include <shastra/front/frontState.h>
#include <shastra/front/frontAppResP.h>

/*static shaFrontAppData frontAppData;*/
shaFrontAppData *pFrontAppData = &frontAppData;
static shastraId frontShastraId;
shastraId       *pFrontSId = &frontShastraId;

static void getFrontShastraIdInfo(Prot0(void));
static char *GetShastraBaseDir();

static void
shastraSetupDefaultResources(pAppData)
    shaFrontAppData *pAppData;
{
    if(pAppData == NULL){
        return;
    }

    /*pAppData->sDirBase = DEFSHASTRABASEDIR;*/
        pAppData->sDirBase = GetShastraBaseDir();
    pAppData->sDirDefs = DEFSHASTRADEFSDIR;
    pAppData->sDirData = DEFSHASTRADATADIR;
    pAppData->sDirBin = DEFSHASTRABINDIR;
    pAppData->sDirHelp = DEFSHASTRAHELPDIR;

    pAppData->sFileHome = DEFSHASTRAHOMEFILE;
    pAppData->sFileLog = DEFSHASTRALOGFILE;
    pAppData->sFileHosts = DEFSHASTRAHOSTSFILE;
    pAppData->sFileUsers = DEFSHASTRAUSERSFILE;
    pAppData->sFileApps = DEFSHASTRAAPPSFILE;
    pAppData->sFileHelp = DEFSHASTRAHELPFILE;

    pAppData->sLocStart = DEFSHASTRASTARTLOCAL;
    pAppData->sRemStart = DEFSHASTRASTARTREMOTE;

    pAppData->sPasswd = DEFSHASTRAPASSWD;

    if(pAppData->fNoGUI){
        pAppData->fConnect = True;
    }
    else{
        pAppData->fConnect = False;
    }
}
```

```
void
shastraFrontSetupApplResDir(sDir)
    char *sDir;
{
  char sbName[1024], *sName;

    if(sDir == NULL){
      sName = resolveNameFromBase(pFrontAppData->sDirBase,
                pFrontAppData->sDirDefs);
    }
    else{
      sName = resolveNameFrom2Bases(pFrontAppData->sDirBase,
                    pFrontAppData->sDirDefs,
                    sDir);
    }
    fprintf(stderr,"getenv()->%s\n",getenv("XAPPLRESDIR"));
    sprintf(sbName,"XAPPLRESDIR=%s", sName);
    fprintf(stderr,"putenv()->%s\n",sbName);
    putenv(sbName);
    fprintf(stderr,"getenv()->%s\n",getenv("XAPPLRESDIR"));
}

void
shastraFrontSetup(argc, argv, sFrName, wgParent, iPort)
    int             argc;
    char          **argv;
    char *sFrName;
    Widget          wgParent;
    int             iPort;
{
  int iStatus;
  int             i;
  int             iSocket;
  extern int      errno;
  char *sName;
  static XrmOptionDescRec xrmOptions[] = {
    DEFSHASTRAXRMOPTIONS
  };

  pFrontAppData = &frontAppData;
  pFrontAppData->pSIdSelf = pFrontSId = &frontShastraId;
  pFrontAppData->wgTop = wgParent;
  pFrontAppData->argc = argc;
  pFrontAppData->argv = argv;
  pFrontAppData->sName = sFrName;
  pFrontAppData->iSvcPort = iPort;
  pFrontAppData->sbMsgBuf = malloc(1024);
  pFrontAppData->fNoGUI = (wgParent == NULL);

  if(wgParent != NULL){
  setupResourceEditHandler(wgParent);
```

```
  XtVaGetApplicationResources(wgParent,
                  (XtPointer)&frontAppData,
                  xrmResources, XtNumber(xrmResources),
                  /*hardcoded non-overridable app resources*/
                  XshaNhelp, False,
                  XshaNusePixmap, False,
                  XshaNservicePort, iPort,
                  NULL);
  /*sanity checking of resources*/
  }
  else{
   /*provide some way of setting resources*/
    shastraSetupDefaultResources(pFrontAppData);
  }
/*
    shastraFrontSetupApplResDir();
*/

  getFrontShastraIdInfo();

  mplexRegisterErrHandler(closedChannelCleanupHandler);

  registerInit();
  kernFrontsInit();
  sesmFrontsInit();
  clientHostsInit();

  if(pFrontAppData->fConnect){
    iStatus = frontKernelConnectReq(pFrontSId);
    for (i = 0; i < 3; i++) {
      if(iStatus == -1){
    if(errno == ECONNREFUSED){
      sName = resolveNameFrom2Bases(pFrontAppData->sDirBase,
                    pFrontAppData->sDirBin,
                    pFrontAppData->sLocStart);
      startShastraKernel(pFrontSId, sName);
      sleep(5);
      iStatus = frontKernelConnectReq(pFrontSId);
    }
    else{
      fprintf(stderr,"Can't start kernel.. Operating standalone!\n");
    }
      }
    }
  }

  clSvrSetSelfModeOprn();
}

void
shastraFrontSetupIcon(sFile)
    char *sFile;
{
```

```
  Pixmap xpmIconBM;
  char sbName[1024], *sName;
  Pixel fg, bg;

  sName = resolveNameFrom2Bases(pFrontAppData->sDirBase,
                pFrontAppData->sDirDefs, sFile);
  xpmIconBM = convertStringToPixmap(pFrontAppData->wgTop, sName);

  if (xpmIconBM != XmUNSPECIFIED_PIXMAP) {
    XtVaSetValues(pFrontAppData->wgTop, XmNiconPixmap, xpmIconBM, NULL);
  }
  /*extend to pixmaps, iconWIndow etc if/when*/
}

void
shastraFrontSetupImage(wg, sFile)
     Widget wg;
     char *sFile;
{
  Pixmap xpmBM;
  char *sName;

  sName = resolveNameFrom2Bases(pFrontAppData->sDirBase,
                pFrontAppData->sDirDefs, sFile);
  xpmBM = convertStringToPixmap(wg, sName);
  if (xpmBM != XmUNSPECIFIED_PIXMAP) {
    XtVaSetValues(wg, XmNlabelPixmap, xpmBM, NULL);
  }
}

static void
getFrontShastraIdInfo()
{
  shastraId      *pSId;
  char *name, nmBuf[256];
  struct hostent *pHostEnt;
  Display *pDisplay;
  uid_t auid;
  struct passwd *apass;
  char *tv;

  pSId = pFrontAppData->pSIdSelf;

  if(pFrontAppData->wgTop != NULL){
  pDisplay = XtDisplay(pFrontAppData->wgTop);
  name = XDisplayString(pDisplay);
  if(name == NULL){
    perror("XDisplayString()");
    strcpy(nmBuf, "anonymous.cs.purdue.edu:0");
  }
  }
  else{
  name = "no-display";
```

```
    }

    pSId->nmDisplay = strdup(name);

  #ifdef SHASTRA4SUN5
    if (sysinfo(SI_HOSTNAME,nmBuf, sizeof(nmBuf)) == -1) {
      perror("sysinfo()");
      strcpy(nmBuf, "anonymous.cs.purdue.edu");
    }
  #else
    if (gethostname(nmBuf, sizeof(nmBuf)) != 0) {
      perror("gethostname()");
      strcpy(nmBuf, "anonymous.cs.purdue.edu");
    }
  #endif
    pSId->nmHost = strdup(nmBuf);

    if ((pHostEnt = gethostbyname(pSId->nmHost)) == NULL) {
      perror("gethostbyname()");
    }
    else{
      unsigned int temp;
      memcpy(&temp, &pHostEnt->h_addr_list[0][0], 4);
      pSId->lIPAddr = ntohl(temp);
      /*pSId->lIPAddr = *(unsigned long *) &pHostEnt->h_addr_list[0][0];*/
    }
  #ifndef DEBUG
    printf("%lu (%lx) -- %s\n", pSId->lIPAddr,
       pSId->lIPAddr, ipaddr2str(pSId->lIPAddr));
  #endif /* DEBUG */

    pSId->iProcId = getpid();

    pSId->iPort = pFrontAppData->iSvcPort;

    pSId->lSIDTag = (pSId->lIPAddr << 16) + pSId->iProcId;
  #ifdef DEBUG
    fprintf(stdout, "SIDTag: %lu\n", pSId->lSIDTag);
  #endif /* DEBUG */

    pSId->dLoadAvg = 0.0;
  #ifdef DEBUG
    fprintf(stdout, "Load : %lf\n", pSId->dLoadAvg);
  #endif /* DEBUG */

    if((pFrontAppData->sName != NULL) && (pFrontAppData->sName[0] != '\0')){
      pSId->nmApplicn = pFrontAppData->sName;
    }
    else if(pFrontAppData->argv[0]){
      if((name = strrchr(pFrontAppData->argv[0], '/')) == NULL){
        name = pFrontAppData->argv[0];
      }
      else{
```

```
      name++;
    }
    pSId->nmApplicn = strdup(name);
  }
  else{
    pSId->nmApplicn = strdup("anonymous");
  }

  auid = getuid();
  apass = getpwuid(auid);
  strcpy(nmBuf, apass->pw_name);
  if (tv =getenv("WEBNAME"))
  {
      pSId->webname = strdup(tv);
      pSId->nmUser = strdup(tv);
  }
  else
  {
      pSId->webname = strdup(nmBuf);
      pSId->nmUser = strdup(nmBuf);
  }

  if(pFrontAppData->sPasswd){
    pSId->nmPasswd = strdup(pFrontAppData->sPasswd);
  }
  else{
    pSId->nmPasswd = strdup(DEFSHASTRAPASSWD);
  }

  if (pFrontAppData->iDbgLevel) {
    outputId(stdout, pSId);
  }
  name = pSId2Str(pSId, PSIDSHOWALL);
  printf("ShastraId: %s\n", name);
  if(name != NULL){
    free(name);
  }
}

shastraId *
getMyFrontShastraId()
{
  if(pFrontAppData){
    return pFrontAppData->pSIdSelf;
  }
  else{
    return NULL;
  }
}

shaFrontAppData *
getMyFrontAppData()
{
```

```
  return pFrontAppData;
}

void            (*clientControlDataFunc) (Prot2(int, shaCmdData**));
void            (*clientOperatorFunc) (Prot1(hostData *));
void            (*clientConnectFunc) (Prot1(hostData *));
void            (*clientSelectFunc) (Prot1(hostData *));
void            (*clientTerminateFunc) (Prot1(hostData *));

void
registerClientControlDataFunc(func)
     void            (*func) ();
{
  clientControlDataFunc = func;
}

void
registerClientTerminateFunc(func)
     void            (*func) ();
{
  clientTerminateFunc = func;
}


void
registerClientSelectFunc(func)
     void            (*func) ();
{
  clientSelectFunc = func;
}

void
registerClientConnectFunc(func)
     void            (*func) ();
{
  clientConnectFunc = func;
}

void
registerClientOperatorFunc(func)
     void            (*func) ();
{
  clientOperatorFunc = func;
}

void (*collabControlDataFunc) (Prot2(int, shaCmdData**));
void (*collabOperatorFunc) (Prot1(hostData *));
void (*collabSelectFunc) (Prot1(hostData *));
void (*collabInitiateFunc) (Prot1(hostData *));
void (*collabTerminateFunc) (Prot1(hostData *));
void (*collabJoinFunc) (Prot1(hostData *));
void (*collabLeaveFunc) (Prot1(hostData *));
void (*collabRemoveFunc) (Prot1(hostData *));
```

```
void (*collabSetPermsFunc) (Prot3(hostData *, shastraIdTag *, unsigned long
    ));
void (*collabGetPermsFunc) (Prot3(hostData *, shastraIdTag *, unsigned long
    ));
void (*collabSetIxnModeFunc) (Prot2(hostData *, unsigned long));
void (*collabGetIxnModeFunc) (Prot2(hostData *, unsigned long));
void (*collabSetFormatFunc) (Prot2(hostData *, unsigned long));
void (*collabGetFormatFunc) (Prot2(hostData *, unsigned long));
void (*collabSetFloorModeFunc) (Prot2(hostData *, unsigned long));
void (*collabGetFloorModeFunc) (Prot2(hostData *, unsigned long));
void (*collabGrabTokenFunc) (Prot2(hostData *, shastraIdTag *));
void (*collabFreeTokenFunc) (Prot2(hostData *, shastraIdTag *));
void (*collabAskTokenFunc) (Prot2(hostData *, shastraIdTag *));
void (*collabTellTokenFunc) (Prot2(hostData *, shastraIdTag *));

void
registerCollabControlDataFunc(func)
     void            (*func) ();
{
  collabControlDataFunc = func;
}

void
registerCollabInitiateFunc(func)
     void            (*func) ();
{
  collabInitiateFunc = func;
}

void
registerCollabTerminateFunc(func)
     void            (*func) ();
{
  collabTerminateFunc = func;
}

void
registerCollabSelectFunc(func)
     void            (*func) ();
{
  collabSelectFunc = func;
}

void
registerCollabJoinFunc(func)
     void            (*func) ();
{
  collabJoinFunc = func;
}

void
registerCollabLeaveFunc(func)
     void            (*func) ();
```

```
{
  collabLeaveFunc = func;
}

void
registerCollabOperatorFunc(func)
     void            (*func) ();
{
  collabOperatorFunc = func;
}

void
registerCollabRemoveFunc(func)
     void            (*func) ();
{
  collabRemoveFunc = func;
}

void
registerCollabSetPermsFunc(func)
     void            (*func) ();
{
  collabSetPermsFunc = func;
}

void
registerCollabGetPermsFunc(func)
     void            (*func) ();
{
  collabGetPermsFunc = func;
}

void
registerCollabSetIxnModeFunc(func)
     void            (*func) ();
{
  collabSetIxnModeFunc = func;
}

void
registerCollabGetIxnModeFunc(func)
     void            (*func) ();
{
  collabGetIxnModeFunc = func;
}

void
registerCollabSetFormatFunc(func)
     void            (*func) ();
{
  collabSetFormatFunc = func;
}
```

```
void
registerCollabGetFormatFunc(func)
     void            (*func) ();
{
  collabGetFormatFunc = func;
}

void
registerCollabSetFloorModeFunc(func)
     void            (*func) ();
{
  collabSetFloorModeFunc = func;
}

void
registerCollabGetFloorModeFunc(func)
     void            (*func) ();
{
  collabGetFloorModeFunc = func;
}

void
registerCollabGrabTokenFunc(func)
     void            (*func) ();
{
  collabGrabTokenFunc = func;
}

void
registerCollabFreeTokenFunc(func)
     void            (*func) ();
{
  collabFreeTokenFunc = func;
}

void
registerCollabAskTokenFunc(func)
     void            (*func) ();
{
  collabAskTokenFunc = func;
}

void
registerCollabTellTokenFunc(func)
     void            (*func) ();
{
  collabTellTokenFunc = func;
}

void (*audioStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*audioEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*audioRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, audioBite *));
void (*audioRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
```

```
void (*audioSendFileFunc) (Prot1(hostData *));
void (*audioSendMsgFunc) (Prot1(hostData *));

void
registerAudioStartFunc(func)
     void            (*func) ();
{
  audioStartFunc = func;
}

void
registerAudioEndFunc(func)
     void            (*func) ();
{
  audioEndFunc = func;
}

void
registerAudioRecvMsgFunc(func)
     void            (*func) ();
{
  audioRecvMsgFunc = func;
}

void
registerAudioRecvFileFunc(func)
     void            (*func) ();
{
  audioRecvFileFunc = func;
}

void
registerAudioSendFileFunc(func)
     void            (*func) ();
{
  audioSendFileFunc = func;
}

void
registerAudioSendMsgFunc(func)
     void            (*func) ();
{
  audioSendMsgFunc = func;
}

void (*videoStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*videoEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*videoRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, videoImg *));
void (*videoRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*videoSendFileFunc) (Prot1(hostData *));
void (*videoSendMsgFunc) (Prot1(hostData *));

void
```

```
registerVideoStartFunc(func)
     void            (*func) ();
{
  videoStartFunc = func;
}

void
registerVideoEndFunc(func)
     void            (*func) ();
{
  videoEndFunc = func;
}

void
registerVideoRecvMsgFunc(func)
     void            (*func) ();
{
  videoRecvMsgFunc = func;
}

void
registerVideoRecvFileFunc(func)
     void            (*func) ();
{
  videoRecvFileFunc = func;
}

void
registerVideoSendFileFunc(func)
     void            (*func) ();
{
  videoSendFileFunc = func;
}

void
registerVideoSendMsgFunc(func)
     void            (*func) ();
{
  videoSendMsgFunc = func;
}

void (*textStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*textEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*textRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*textRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*textSendFileFunc) (Prot1(hostData *));
void (*textSendMsgFunc) (Prot1(hostData *));

void
registerTextStartFunc(func)
     void            (*func) ();
{
  textStartFunc = func;
```

```
}

void
registerTextEndFunc(func)
     void            (*func) ();
{
  textEndFunc = func;
}

void
registerTextRecvMsgFunc(func)
     void            (*func) ();
{
  textRecvMsgFunc = func;
}

void
registerTextRecvFileFunc(func)
     void            (*func) ();
{
  textRecvFileFunc = func;
}

void
registerTextSendFileFunc(func)
     void            (*func) ();
{
  textSendFileFunc = func;
}

void
registerTextSendMsgFunc(func)
     void            (*func) ();
{
  textSendMsgFunc = func;
}

void (*pictStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*pictEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*pictRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, pictPieces *));
void (*pictRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*pictSendFileFunc) (Prot1(hostData *));
void (*pictSendMsgFunc) (Prot1(hostData *));

void
registerPictStartFunc(func)
     void            (*func) ();
{
  pictStartFunc = func;
}

void
registerPictEndFunc(func)
```

```
     void            (*func) ();
{
  pictEndFunc = func;
}

void
registerPictRecvMsgFunc(func)
     void            (*func) ();
{
  pictRecvMsgFunc = func;
}

void
registerPictRecvFileFunc(func)
     void            (*func) ();
{
  pictRecvFileFunc = func;
}

void
registerPictSendFileFunc(func)
     void            (*func) ();
{
  pictSendFileFunc = func;
}

void
registerPictSendMsgFunc(func)
     void            (*func) ();
{
  pictSendMsgFunc = func;
}

void (*xsCntlStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*xsCntlEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*xsCntlRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, xsCntlDatas *)
    );
void (*xsCntlRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*xsCntlSendFileFunc) (Prot1(hostData *));
void (*xsCntlSendMsgFunc) (Prot1(hostData *));

void
registerXSCntlStartFunc(func)
     void            (*func) ();
{
  xsCntlStartFunc = func;
}

void
registerXSCntlEndFunc(func)
     void            (*func) ();
{
  xsCntlEndFunc = func;
```

```
}

void
registerXSCntlRecvMsgFunc(func)
     void            (*func) ();
{
  xsCntlRecvMsgFunc = func;
}

void
registerXSCntlRecvFileFunc(func)
     void            (*func) ();
{
  xsCntlRecvFileFunc = func;
}

void
registerXSCntlSendFileFunc(func)
     void            (*func) ();
{
  xsCntlSendFileFunc = func;
}

void
registerXSCntlSendMsgFunc(func)
     void            (*func) ();
{
  xsCntlSendMsgFunc = func;
}

void (*polyStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*polyEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*polyRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, ipimageData *));
void (*polyRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*polySendFileFunc) (Prot1(hostData *));
void (*polySendMsgFunc) (Prot1(hostData *));

void
registerPolyStartFunc(func)
     void            (*func) ();
{
  polyStartFunc = func;
}

void
registerPolyEndFunc(func)
     void            (*func) ();
{
  polyEndFunc = func;
}

void
registerPolyRecvMsgFunc(func)
```

```
     void            (*func) ();
{
  polyRecvMsgFunc = func;
}

void
registerPolyRecvFileFunc(func)
     void            (*func) ();
{
  polyRecvFileFunc = func;
}

void
registerPolySendFileFunc(func)
     void            (*func) ();
{
  polySendFileFunc = func;
}

void
registerPolySendMsgFunc(func)
     void            (*func) ();
{
  polySendMsgFunc = func;
}

void (*pntrStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*pntrEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*pntrRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, shaDoubles *));
void (*pntrRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*pntrSendFileFunc) (Prot1(hostData *));
void (*pntrSendMsgFunc) (Prot1(hostData *));

void
registerPntrStartFunc(func)
     void            (*func) ();
{
  pntrStartFunc = func;
}

void
registerPntrEndFunc(func)
     void            (*func) ();
{
  pntrEndFunc = func;
}

void
registerPntrRecvMsgFunc(func)
     void            (*func) ();
{
  pntrRecvMsgFunc = func;
}
```

```
void
registerPntrRecvFileFunc(func)
     void            (*func) ();
{
  pntrRecvFileFunc = func;
}

void
registerPntrSendFileFunc(func)
     void            (*func) ();
{
  pntrSendFileFunc = func;
}

void
registerPntrSendMsgFunc(func)
     void            (*func) ();
{
  pntrSendMsgFunc = func;
}

void (*cursorStartFunc) (Prot2(hostData *, shastraIdTag *));
void (*cursorEndFunc) (Prot2(hostData *, shastraIdTag *));
void (*cursorRecvMsgFunc) (Prot3(hostData *, shastraIdTag *, shaDoubles *))
    ;
void (*cursorRecvFileFunc) (Prot3(hostData *, shastraIdTag *, char *));
void (*cursorSendFileFunc) (Prot1(hostData *));
void (*cursorSendMsgFunc) (Prot1(hostData *));

void
registerCursorStartFunc(func)
     void            (*func) ();
{
  cursorStartFunc = func;
}

void
registerCursorEndFunc(func)
     void            (*func) ();
{
  cursorEndFunc = func;
}

void
registerCursorRecvMsgFunc(func)
     void            (*func) ();
{
  cursorRecvMsgFunc = func;
}

void
registerCursorRecvFileFunc(func)
```

```
     void            (*func) ();
{
  cursorRecvFileFunc = func;
}

void
registerCursorSendFileFunc(func)
     void            (*func) ();
{
  cursorSendFileFunc = func;
}

void
registerCursorSendMsgFunc(func)
     void            (*func) ();
{
  cursorSendMsgFunc = func;
}

static char *GetShastraBaseDir()
{
    char *dname;

    if (dname = getenv("SHASTRADIR"))
    {
         return(dname);
    }
    else
    {
        dname = strdup(DEFSHASTRABASEDIR);
    }
    return(dname);
}
```

```
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
/**
    **/
/** This SHASTRA software is not in the Public Domain. It is distributed on
    **/
/** a person to person basis, solely for educational use and permission is
    **/
/** NOT granted for its transfer to anyone or for its use in any commercial
    **/
/** product.  There is NO warranty on the available software and neither
    **/
/** Purdue University nor the Applied Algebra and Geometry group directed
    **/
/** by C.  Bajaj accept responsibility for the consequences of its use.
    **/
/**
    **/
/*****************************************************************************
    ***/
/*****************************************************************************
    ***/
#include <stdio.h>
#include <sys/errno.h>
#include <netdb.h>

#include <shastra/shastra.h>

#include <shastra/utils/list.h>

#include <shastra/datacomm/shastraIdH.h>
#include <shastra/datacomm/shastraIdTagH.h>
#include <shastra/datacomm/shastraDataH.h>

#include <shastra/shautils/shautils.h>
#include <shastra/shautils/kernelFrontsP.h>
#include <shastra/shautils/kernelFronts.h>
#include <shastra/shautils/sesMgrFronts.h>
#include <shastra/shautils/clientHosts.h>

#include <shastra/network/server.h>
#include <shastra/network/mplex.h>
#include <shastra/network/hostMgr.h>

#include <shastra/front/shastraCntl.h>
#include <shastra/front/clSvrCntl.h>
#include <shastra/front/frontP.h>
#include <shastra/front/front.h>
#include <shastra/front/frontState.h>
#include <shastra/front/front_client.h>
#include <shastra/front/front_clientP.h>
```

```
#include <shastra/front/frontCollClient.h>

shaCmdData      frontCmdData;

cmCommand       frontCmdTab[] = FRONT_CLIENTCMDS;
#define FRONT_NCMDS (sizeof(frontCmdTab)/sizeof(cmCommand))
int             frontNCmds = FRONT_NCMDS;

cmCommand       frontInCmdTab[] = FRONT_CLIENTINCMDS;
#define FRONT_INNCMDS (sizeof(frontInCmdTab)/sizeof(cmCommand))
int             frontInNCmds = FRONT_INNCMDS;

hostData       *pHostKernel;

#define checkConn() \
     if ((pHostKernel == NULL) || \
     (pHostKernel->fStatus == shaError)) \
{ \
  fprintf(stderr,"Connection to Shastra is bad!\n");    \
  return -1; \
}

#define sendReqString(s, arg) \
    if(hostSendQueuedRequest(pHostKernel, s, arg) == -1) \
{ \
  pHostKernel->fStatus = shaError; \
  closedChannelCleanupHandler(pHostKernel->fdSocket); \
  fprintf(stderr,"Error in Sending Shastra Operation Request\n"); \
  return -1; \
}

#define sendDataString(s) \
     if(cmSendString(pHostKernel->fdSocket, s) == -1) \
{ \
  pHostKernel->fStatus = shaError; \
  closedChannelCleanupHandler(pHostKernel->fdSocket); \
  fprintf(stderr,"Error in Sending Shastra Operation Data\n"); \
  return -1; \
}


#define ShastraIdIn(filedesc, pShaId) \
     if(shastraIdIn(pHostKernel->fdSocket, pShaId) == -1) \
{\
  pHostKernel->fStatus = shaError;\
  closedChannelCleanupHandler(pHostKernel->fdSocket);\
  fprintf(stderr, "Error Receiving SID from Kernel\n");\
  return -1;\
}

#define ShastraIdOut(filedesc, pShaId)\
     if(shastraIdOut(pHostKernel->fdSocket, pShaId) == -1)\
{\
```

```
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Sending SID to Kernel\n");\
 return -1;\
}

#define ShastraIdsIn(filedesc, pShaIds)\
     if(shastraIdsIn(pHostKernel->fdSocket, pShaIds) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Receiving SIDs from Kernel\n");\
 return -1;\
}

#define ShastraIdsOut(filedesc, pShaIds)\
     if(shastraIdsOut(pHostKernel->fdSocket, pShaIds) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Sending SIDs to Kernel\n");\
 return -1;\
}

#define ShastraIdTagIn(filedesc, pShaIdTag)\
     if(shastraIdTagIn(pHostKernel->fdSocket, pShaIdTag) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Receiving SIDTag from Kernel\n");\
 return -1;\
}

#define ShastraIdTagOut(filedesc, pShaIdTag)\
     if(shastraIdTagOut(pHostKernel->fdSocket, pShaIdTag) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Sending SIDTag to Kernel\n");\
 return -1;\
}

#define ShastraIdTagsIn(filedesc, pShaIdTags)\
     if(shastraIdTagsIn(pHostKernel->fdSocket, pShaIdTags) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Receiving SIDTags from Kernel\n");\
 return -1;\
}

#define ShastraIdTagsOut(filedesc, pShaIdTags)\
     if(shastraIdTagsOut(pHostKernel->fdSocket, pShaIdTags) == -1)\
```

```
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Sending SIDTags to Kernel\n");\
 return -1;\
}

#define ShastraULongIn(filedesc, pULong)\
     if(shaULongIn(pHostKernel->fdSocket, pULong) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Receiving pULong from Kernel\n");\
 return -1;\
}

#define ShastraULongOut(filedesc, pULong)\
     if(shaULongOut(pHostKernel->fdSocket, pULong) == -1)\
{\
 pHostKernel->fStatus = shaError;\
 closedChannelCleanupHandler(pHostKernel->fdSocket);\
 fprintf(stderr, "Error Sending pULong to Kernel\n");\
 return -1;\
}


/*
 * Function
 */
int
startSystemReq(pHostKr, pCreateSId)
     hostData       *pHostKr;
     shastraId *pCreateSId;
{
  char *sName;
  shastraId       *pSId;
  shastraIdTag *pSIdTag;
  struct hostent *pHostEnt;
  int krIndex;
  int fMine = 0;
  unsigned int temp;

  if(pCreateSId == NULL){
    fprintf(stderr,"startSystemReq()-> bad args!\n");
    return -1;
  }
  if ((pHostEnt = gethostbyname(pCreateSId->nmHost)) == NULL) {
    perror("gethostbyname()");
    sprintf(pFrontAppData->sbMsgBuf, "Bad Host %s\n", pCreateSId->nmHost);
    showShastraInfo(pFrontAppData->sbMsgBuf);
    return -1;
  }
  memcpy(&temp, &pHostEnt->h_addr_list[0][0], 4);
```

```
  pCreateSId->lIPAddr = ntohl(temp);
  printf("%lu -- %s\n", pCreateSId->lIPAddr,
     ipaddr2str(pCreateSId->lIPAddr));

  pCreateSId->lSIDTag = 0;
  pCreateSId->dLoadAvg = 0;
  pCreateSId->nmUser = pFrontSId->nmUser;
  pCreateSId->iPort = 0;
  pCreateSId->iProcId = 0;

  if (pFrontAppData->iDbgLevel) {
    outputId(stdout, pCreateSId);
  }
  if ((krIndex = locateByNameKernFronts(pCreateSId)) != -1) {
    pSIdTag = KernFrontSIdTag(krIndex);
    pSId = getSIdByTagInSIds(pSIdTag, &shastraKernIds);
    if(!strcmp(pSId->nmUser, pFrontSId->nmUser)){
      fMine = 1;
    }
  }
/*CHECK -- force rsh*/
  if(fMine || !fMine){
    sName = resolveNameFrom2Bases(pFrontAppData->sDirBase,
                  pFrontAppData->sDirBin,
                  pFrontAppData->sRemStart);
    startShastraSystem(pCreateSId, sName);
  }
  else{
    checkConn();
    sendReqString(REQ_START_SYSTEM, NULL);
    ShastraIdOut(pHostKernel->fdSocket, pCreateSId);
    cmFlush(pHostKernel->fdSocket);
  }
  return 0;
}

/*
 * Function
 */
int
startSystemRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_START_SYSTEM);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
endSystemReq(pHostKr, pSId)
```

```
     hostData       *pHostKr;
     shastraId      *pSId;
{
  if(pSId == NULL){
    fprintf(stderr,"endSystemReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_END_SYSTEM, NULL);
  ShastraIdOut(pHostKernel->fdSocket, pSId);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
endSystemRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_END_SYSTEM);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
connectSystemReq(pHostKr, pSId)
     hostData       *pHostKr;
     shastraId *pSId;
{
  if(pSId == NULL){
    fprintf(stderr,"connectSystemReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_CONNECT_SYSTEM, NULL);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
connectSystemRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_CONNECT_SYSTEM);
  showShastraInfo(pFrontAppData->sbMsgBuf);
```

```
  return 0;
}

/*
 * Function
 */
int
setShastraIdReq(pHostKr, pSId)
     hostData       *pHostKr;
     shastraId *pSId;
{
  if(pSId == NULL){
    fprintf(stderr,"setShastraIdReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_SET_SHASTRAID, NULL);
  ShastraIdOut(pHostKernel->fdSocket, pSId);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
setShastraIdRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_SET_SHASTRAID);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
getShaKernIdReq(pHostKr)
     hostData       *pHostKr;
{
  checkConn();
  sendReqString(REQ_GET_SHAKERNID, NULL);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
getShaKernIdRespHandler(fd)
     int             fd;
```

```
{

  ShastraIdsIn(fd, &shastraKernIds);
  if (pFrontAppData->iDbgLevel) {
    outputIds(stderr, &shastraKernIds);
  }
  adjustKrFrMapSize(shastraKernIds.shastraIds_len);
  updateKrFrMap(&shastraKernIds);

  setKernelNamesOprn();
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNID);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
getShaKernFrIdReq(pHostKr, pSId)
     hostData       *pHostKr;
     shastraId      *pSId;
{
  if(pSId == NULL){
    fprintf(stderr,"getShaKernFrIdReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_GET_SHAKERNFRID, NULL);
  ShastraIdOut(pHostKernel->fdSocket, pSId);

  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
getShaKernFrIdRespHandler(fd)
     int             fd;
{
  int             iObjIndex;
  static shastraId inShaId;
  static shastraIds inShaIds;
  shastraIds     *pSIds;
  int             krIndex;

fprintf(stderr, "Should be getting front Id's!\n");
  ShastraIdIn(fd, &inShaId);
  krIndex = locateKernFronts(&inShaId);
  if (krIndex == -1) {
```

```
    krIndex = occupyKrFrFreeSlot(&inShaId);
  }
  pSIds = getKernFrontSIds(&inShaId);
  ShastraIdsIn(fd, pSIds);
  if (pFrontAppData->iDbgLevel) {
    outputIds(stderr, pSIds);
  }
  setKernelFrontNamesOprn(inShaId.lSIDTag);
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SHAKERNFRID);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
getShaSesmIdReq(pHostKr)
     hostData       *pHostKr;
{
  checkConn();
  sendReqString(REQ_GET_SHASESMID, NULL);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
getShaSesmIdRespHandler(fd)
     int             fd;
{

  ShastraIdsIn(fd, &shastraSesmIds);
  if (pFrontAppData->iDbgLevel) {
    outputIds(stderr, &shastraSesmIds);
  }
  adjustSmFrMapSize(shastraSesmIds.shastraIds_len);
  updateSmFrMap(&shastraSesmIds);

  setSesMgrNamesOprn();
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SHASESMID);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
```

```
getShaSesmFrIdReq(pHostKr, pSIdTag)
     hostData       *pHostKr;
     shastraIdTag   *pSIdTag;
{
  if(pSIdTag == NULL){
    fprintf(stderr,"getShaSesmFrIdReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_GET_SHASESMFRID, (char *) NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}


/*
 * Function
 */
int
getShaSesmFrIdRespHandler(fd)
     int             fd;
{
  int             smIndex;
  shastraIdTag    inShaIdTag;
  static shastraIdTags inShaIdTags;
  static shastraIdTags inShaPermTags;
  shastraIdTags  *pSIdTags;
  shastraIdTags  *pPermTags;

  ShastraIdTagIn(fd, &inShaIdTag);
  smIndex = locateSesmFronts(&inShaIdTag);
  if (smIndex == -1) {
    fprintf(stderr, "getShaSesmFrIdRespHandler()->can't locate sesMgr!\n");
    ShastraIdTagsIn(fd, &inShaIdTags);
    ShastraIdTagsIn(fd, &inShaPermTags);
    return -1;
  }
  pSIdTags = getSesmFrontSIdTags(&inShaIdTag);
  ShastraIdTagsIn(fd, pSIdTags);
  pPermTags = getSesmFrontPermTags(&inShaIdTag);
  ShastraIdTagsIn(fd, pPermTags);
  if (pFrontAppData->iDbgLevel) {
    outputIdTags(stderr, pSIdTags);
    outputIdTags(stderr, pPermTags);
  }
  updateSmFrMap(&shastraSesmIds);
  setSesMgrFrontNamesOprn(inShaIdTag);
  setCollabFrontPermsOprn(inShaIdTag);
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_GET_SHASESMFRID);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}
```

```
/*
 * Function
 */
int
helpReq(pHostKr)
     hostData       *pHostKr;
{
  checkConn();
  sendReqString(REQ_HELP, NULL);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int
helpRespHandler(fd)
     int             fd;
{
  standardHelpRespHandler(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_HELP);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
shaFrontQuitReq(pHostKr)
     hostData       *pHostKr;
{
  if ((pHostKernel != NULL) && (pHostKernel->fStatus != shaError)) {
    sendReqString(REQ_QUIT, NULL);
    cmFlush(pHostKernel->fdSocket);
    quitRespHandler(pHostKernel->fdSocket);
  }
  else{
    quitRespHandler(0);
  }

  return 0;
}

/*
 * Function
 */
int
quitRespHandler(fd)
     int             fd;
```

```
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_QUIT);
  showShastraInfo(pFrontAppData->sbMsgBuf);

  if(pHostKernel){
    fd = pHostKernel->fdSocket;
  }
  mplexUnRegisterChannel(fd);
  XtDestroyApplicationContext(
                  XtWidgetToApplicationContext(pFrontAppData->wgTop));
  exit(0);
  return 0;
}


/*
 * Function
 */
int
clntConnectReq(pHostKr, pSId, pCmdData)
     hostData       *pHostKr;
     shastraId      *pSId;
     shaCmdData     *pCmdData;
{
  int             status;
  int             clntSocket;
  hostData       *pHost;

  if((pSId == NULL) || (pCmdData == NULL)){
    fprintf(stderr,"clntConnectReq()-> bad args!\n");
    return -1;
  }
  if(pCmdData == NULL){
    fprintf(stderr,"clntConnectReq()-> Warning.. No Control Data!\n");
  }
  if (pFrontAppData->iDbgLevel) {
    outputId(stdout, pSId);
  }
  status = cmClientConnect2Server(pSId->nmHost, pSId->nmApplicn,
                  pSId->iPort, &clntSocket);
  if (status == -1) {
    sprintf(pFrontAppData->sbMsgBuf, "clientConnectReq()-- Couldn't connect
        \n");
    showShastraInfo(pFrontAppData->sbMsgBuf);
    return -1;
  } else {
    sprintf(pFrontAppData->sbMsgBuf, "clientConnectReq()-- connected\n");
    showShastraInfo(pFrontAppData->sbMsgBuf);
  }

  shaKernFlags[clntSocket] = SHAFRONT;
  pHost = (hostData *) malloc(sizeof(hostData));
  memset(pHost, 0, sizeof(hostData));
```

```
  pHost->fdSocket = clntSocket;
  pHost->lSIDTag = pSId->lSIDTag;
  pHost->pSId = copyId(pSId, NULL);
  pHost->sendList = listMakeNew();
  pHost->recvList = listMakeNew();
  pHost->fStatus = shaWait2Send;

  if (locateClientHosts(pSId) == -1) {
    occupyClHostFreeSlot(pSId);
  }
  updateAddClHost(pSId, pHost);

  if (clientConnectFunc != NULL) {
    (*clientConnectFunc) (pHost);
  } else {
    showShastraInfo("clntConnectReq() -- Error! No handler");
  }
  setClSvrServerNamesOprn(pSId);
  clSvrSetCurrHostOprn(pHost, False);
  if (mplexRegisterChannel(pHost->fdSocket, shaClientHandler,
              pCmdData, (char *) pHost) == -1) {
    fprintf(stderr, "clntConnectReq()->Couldn't Register Client Handler!!\
        n");
    pHost->fStatus = shaError;
    return -1;
  }
  mplexSetHostData(pHost->fdSocket, pHost);
  if((pHostKr = mplexGetHostData(pHost->fdSocket)) != pHost){
    fprintf(stderr,"clntConnectReq()->mplexSetHostData problem!\n");
  }
  /*
  cmJoinCmdData(&frontCollCmdData, pCmdData); join to standard client
      handling
  */
  return 0;
}


/*
 * Function
 */
int
clntTerminateReq(pHostKr, pHost)
     hostData       *pHostKr;
     hostData       *pHost;
{

  if(pHost == NULL){
    fprintf(stderr, "clntTerminateReq()->Bad Args!\n");
    return -1;
  }
  clntDisconnectHandler(pHost->fdSocket);
  mplexUnRegisterChannel(pHost->fdSocket);
```

```
/*CHECK technically, should send quit req, and response should do all this*
    /
}

int
clntDisconnectHandler(fd)
     int fd;
{
  hostData       *pHost;
  shastraId      *pSIdSvr;
  shastraIdTag      *pSIdTag;

  pHost = mplexGetHostData(fd);
  pSIdTag = &pHost->lSIDTag;
  pSIdSvr = pHost->pSId;
  if (pSIdSvr == NULL) {
    fprintf(stderr, "clntDisconnectHandler()->Missing Host System!\n");
    return -1;
  }
  updateRmvClHostByIdTag(pSIdSvr, pSIdTag);
  setClSvrServerNamesOprn(pSIdSvr);
  clSvrResetCurrHostOprn(pSIdSvr, True);
  if (clientTerminateFunc != NULL) {
    (*clientTerminateFunc) (pHost);
  } else {
    showShastraInfo("clntDisconnectRespHandler() -- Error! No handler\n");
  }
/*
  free(pHost->pSId, free(pHost);
  */
}


/*
 * Function
 */
int
clntTerminateServerReq(pHostKr, pHost)
     hostData       *pHostKr;
     hostData       *pHost;
{

  if(pHost == NULL){
    fprintf(stderr, "clntTerminateServerReq()->Bad Args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_TERMINATE, NULL);
  cmFlush(pHost->fdSocket);
  return 0;
  clntDisconnectHandler(pHost->fdSocket);
  mplexUnRegisterChannel(pHost->fdSocket);
/*CHECK technically, should send quit req, and response should do all this*
```

```
    /
}

/*
 * Function
 */
int
collInitiateReq(pHostKr, pSIdTags, perms, lIdTag)
     hostData       *pHostKr;
     shastraIdTags  *pSIdTags;
     unsigned long   perms, lIdTag;
{
  if(pSIdTags == NULL){
    fprintf(stderr,"collInitiateReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_INITIATE, NULL);
  ShastraIdTagsOut(pHostKernel->fdSocket, pSIdTags);
  ShastraULongOut(pHostKernel->fdSocket, &perms);
  ShastraULongOut(pHostKernel->fdSocket, &lIdTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
collInitiateRespHandler(fd)
     int             fd;
{
  if (collabInitiateFunc != NULL) {
    (*collabInitiateFunc) (NULL);
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_INITIATE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collAutoInitiateReq(pHostKr, pSIdTags, perms, lIdTag)
     hostData       *pHostKr;
     shastraIdTags  *pSIdTags;
     unsigned long   perms, lIdTag;
{
  if(pSIdTags == NULL){
    fprintf(stderr,"collAutoInitiateReq()-> bad args!\n");
    return -1;
  }
  checkConn();
```

```
  sendReqString(REQ_COLL_AUTOINITIATE, NULL);
  ShastraIdTagsOut(pHostKernel->fdSocket, pSIdTags);
  ShastraULongOut(pHostKernel->fdSocket, &perms);
  ShastraULongOut(pHostKernel->fdSocket, &lIdTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
collAutoInitiateRespHandler(fd)
     int             fd;
{
  if (collabInitiateFunc != NULL) {
    (*collabInitiateFunc) (NULL);
  }
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_AUTOINITIATE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
collAskJoinReq(pHostKr, pSmSIdTag, pSIdTag)
     hostData       *pHostKr;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pSIdTag;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL)){
    fprintf(stderr,"collAskJoinReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_ASKJOIN, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
collAskJoinRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOIN);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
```

```
}

/*
 * Function
 */
int
collInviteJoinReq(pHostKr, pSmSIdTag, pSIdTag, pLdrSIdTag, pPermTag)
     hostData       *pHostKr;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pSIdTag;
     shastraIdTag   *pLdrSIdTag;
     shastraIdTag   *pPermTag;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pLdrSIdTag == NULL)
     || (pPermTag == NULL)){
    fprintf(stderr,"collInviteJoinReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_INVITEJOIN, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pLdrSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pPermTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
collInviteJoinRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_INVITEJOIN);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
collTellJoinReq(pHostKr, pSmSIdTag, pSIdTag, pPermTag)
     hostData       *pHostKr;
     shastraIdTag   *pSmSIdTag;
     shastraIdTag   *pSIdTag;
     shastraIdTag   *pPermTag;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pPermTag == NULL)){
    fprintf(stderr,"collTellJoinReq()-> bad args!\n");
    return -1;
  }
```

```
  checkConn();
  sendReqString(REQ_COLL_TELLJOIN, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pPermTag);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}


/*
 * Function
 */
int
collTellJoinRespHandler(fd)
     int             fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_TELLJOIN);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
collInviteRespHandler(fd)
     int             fd;
{
  shastraIdTag    sesmSIdTag;
  shastraIdTag    frontSIdTag;
  shastraIdTag    leaderSIdTag;
  shastraIdTag    frontPermTag;

  ShastraIdTagIn(fd, &sesmSIdTag);
  ShastraIdTagIn(fd, &frontSIdTag);
  ShastraIdTagIn(fd, &leaderSIdTag);
  ShastraIdTagIn(fd, &frontPermTag);

/*
  collJoinPromptOprn(sesmSIdTag, leaderSIdTag, frontPermTag);
  */
  collabInvitePromptOprn(sesmSIdTag, leaderSIdTag, frontPermTag);

  sprintf(pFrontAppData->sbMsgBuf, "Done (end)-- %s\n", REQ_COLL_INVITEJOIN
      );
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int
```

```
collTellJnRespHandler(fd)
     int             fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    sIdTag;
  shastraIdTag    permTag;
  shastraId      *pSId;
  shaCmdData *pCmdData = NULL;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  ShastraIdTagIn(fd, &permTag);
  pSId = mapSIdTag2SId(&smSIdTag);
  if(collabControlDataFunc){
    (*collabControlDataFunc)(shastraNameToService(pSId->nmApplicn), &
        pCmdData);
    if(pCmdData == NULL){
      fprintf(stderr,"collTellJnRespHandler()->Invalid Control Data!\n");
    }
  }
  else{
    fprintf(stderr,"collTellJnRespHandler()->Can't Obtain Control Data!\n")
        ;
  }

  collJoinReq((hostData*)NULL, pSId, &permTag, pCmdData);
  sprintf(pFrontAppData->sbMsgBuf, "Done (end) -- %s\n", REQ_COLL_TELLJOIN)
      ;
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}


/*
 * Function
 */
int
terminateHandler(i)
     int             i;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_TERMINATE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  shaFrontQuitReq(pHostKernel);
  return 0;
}
/*
 * Function
 */
int
closedChannelCleanupHandler(fd)
     int             fd;
{
  hostData       *pHost;
```

```
  pHost = mplexGetHostData(fd);
  if (pHost == NULL) {
    fprintf(stderr, "closedChannelCleanupHandler(%d)->NULL Host data!\n",
        fd);
  }
  else{
    if (shaKernFlags[fd] == SHAKERNEL) {
      fprintf(stderr, "closedChannelCleanupHandler(%d)->Kernel Disconnected
          !\n",
        fd);
      kernelDisconnectHandler(fd);
    } else if (shaKernFlags[fd] == SHASESMGR) {
      collLeaveRespHandler(fd);
    } else if (shaKernFlags[fd] == SHAFRONT) {
      clntDisconnectHandler(fd);
    }
  }
  mplexUnRegisterChannel(fd);
  return 0;
}

/*
 * Function
 */
int collInviteMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"collInviteMsgReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_INVITEMSG, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int collInviteMsgRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_INVITEMSG);
```

```
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collInviteMsgHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabRecvdInviteMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COLL_INVITEMSG)
      ;
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int collInvRespMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"collInviteJoinReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_INVRESPMSG, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
```

```
 */
int collInvRespMsgRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_INVRESPMSG);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collInvRespMsgHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabRecvdInvRespMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COLL_INVRESPMSG
      );
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int collInviteStatusReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     shaULong lStatus;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"collInviteStatusReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_INVITESTATUS, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  ShastraULongOut(pHostKernel->fdSocket, &lStatus);
  cmFlush(pHostKernel->fdSocket);
```

```
  return 0;
}

/*
 * Function
 */
int collInviteStatusRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_INVITESTATUS);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collInviteStatusHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  shaULong      lStatus;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  ShastraULongIn(fd, &lStatus);
  collabShowInviteStatusOprn(smSIdTag, toSIdTag, sIdTag, lStatus);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COLL_INVITESTATUS);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collAskJoinMsgReq(pHostKr, pSmSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL)){
    fprintf(stderr,"collAskJoinMsgReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_ASKJOINMSG, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
```

```
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int collAskJoinMsgRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_ASKJOINMSG);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collAskJoinMsgHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabRecvdAskJoinMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COLL_ASKJOINMSG
      );
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int collAskJnRespMsgReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"collAskJnRespMsgReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_ASKJNRESPMSG, NULL);
```

```
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int collAskJnRespMsgRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNRESPMSG);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collAskJnRespMsgHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  collabRecvdAskJnRespMessageOprn(smSIdTag, sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COLL_ASKJNRESPMSG);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int collAskJnStatusReq(pHostKr, pSmSIdTag, pToSIdTag, pSIdTag, lStatus)
     hostData* pHostKr;
     shastraIdTag *pSmSIdTag;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     shaULong lStatus;
{
  if((pSmSIdTag == NULL) || (pSIdTag == NULL) || (pToSIdTag == NULL)){
```

```
    fprintf(stderr,"collAskJnStatusReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COLL_ASKJNSTATUS, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pSmSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  ShastraULongOut(pHostKernel->fdSocket, &lStatus);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int collAskJnStatusRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COLL_ASKJNSTATUS);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int collAskJnStatusHandler(fd)
     int fd;
{
  shastraIdTag    smSIdTag;
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  shaULong      lStatus;

  ShastraIdTagIn(fd, &smSIdTag);
  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  ShastraULongIn(fd, &lStatus);
  collabShowAskJoinStatusOprn(smSIdTag, toSIdTag, sIdTag, lStatus);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COLL_ASKJNSTATUS);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgTextReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
```

```
     char *sbMsg;
{
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"collInviteJoinReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGTEXT, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgTextRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXT);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgTextHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  shastraRecvdMessageOprn(sIdTag, sMsg);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGTEXT);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int commMsgTextFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
```

```
     char *sbMsg;
{
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"commMsgTextFileReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGTEXTFILE, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgTxtFileRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGTEXTFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgTxtFileHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGTEXTFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int commMsgAudioReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
```

```
     char *sbMsg;
{
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"commMsgAudioReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGAUDIO, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgAudioRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIO);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgAudioHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGAUDIO);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int commMsgAudioFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
```

```
{
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"commMsgAudioFileReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGAUDIOFILE, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgAudioFileRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGAUDIOFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgAudioFileHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGAUDIOFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int commMsgVideoReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
```

```
{
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"commMsgVideoReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGVIDEO, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgVideoRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEO);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgVideoHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n", REQ_COMM_MSGVIDEO);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}

/*
 * Function
 */
int commMsgVideoFileReq(pHostKr, pToSIdTag, pSIdTag, sbMsg)
     hostData* pHostKr;
     shastraIdTag *pToSIdTag;
     shastraIdTag *pSIdTag;
     char *sbMsg;
{
```

```
  if((pSIdTag == NULL) || (pToSIdTag == NULL)){
    fprintf(stderr,"commMsgVideoFileReq()-> bad args!\n");
    return -1;
  }
  checkConn();
  sendReqString(REQ_COMM_MSGVIDEOFILE, NULL);
  ShastraIdTagOut(pHostKernel->fdSocket, pToSIdTag);
  ShastraIdTagOut(pHostKernel->fdSocket, pSIdTag);
  sendDataString(sbMsg);
  cmFlush(pHostKernel->fdSocket);
  return 0;
}

/*
 * Function
 */
int commMsgVideoFileRespHandler(fd)
     int fd;
{
  sprintf(pFrontAppData->sbMsgBuf, "Done -- %s\n", REQ_COMM_MSGVIDEOFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  return 0;
}

/*
 * Function
 */
int commMsgVideoFileHandler(fd)
     int fd;
{
  shastraIdTag    toSIdTag;
  shastraIdTag    sIdTag;
  char *sMsg;

  ShastraIdTagIn(fd, &toSIdTag);
  ShastraIdTagIn(fd, &sIdTag);
  sMsg = cmReceiveString(fd);
  sprintf(pFrontAppData->sbMsgBuf, "Done (in) -- %s\n",
      REQ_COMM_MSGVIDEOFILE);
  showShastraInfo(pFrontAppData->sbMsgBuf);
  free(sMsg);
  return 0;
}


int
frontKernelConnectReq(pSId)
     shastraId *pSId;
{
  int iStatus, iSocket;
  hostData *pHost;

  if(pHostKernel){
```

```
    return 0;
  }
  if(pSId == NULL){
    pSId = pFrontSId;
  }
  iStatus = cmClientConnect2Server(pSId->nmHost, SHASTRA_SERVICE_NAME, 0,
                  &iSocket);
  if (iStatus == -1){
    return -1;
  }

  frontCmdData.pCmdTab = frontCmdTab;
  frontCmdData.nCmds = frontNCmds;
  frontCmdData.pCmdTabIn = frontInCmdTab;
  frontCmdData.nCmdsIn = frontInNCmds;

  if (mplexRegisterChannel(iSocket, shaClientHandler,
              &frontCmdData, (char *) pHostKernel) == -1) {
    fprintf(stderr, "mplexRegisterChannel()->Error!\n");
    close(iSocket);
    return -1;
  }
  pHostKernel = (hostData*)malloc(sizeof(hostData));
  memset(pHostKernel, 0, sizeof(hostData));

  shaKernFlags[iSocket] = SHAKERNEL;
  pHostKernel->fdSocket = iSocket;
  pHostKernel->sendList = listMakeNew();
  pHostKernel->recvList = listMakeNew();
  pHostKernel->fStatus = shaWait2Send;

  mplexSetHostData(pHostKernel->fdSocket, pHostKernel);
  if((pHost = mplexGetHostData(pHostKernel->fdSocket)) != pHostKernel){
    fprintf(stderr,"frontKernelConnectReq()->mplexSetHostData problem!\n");
  }

  setShastraIdReq(pHostKernel, pSId);
  return 0;
}

int
frontKernelDisconnectReq(pSId)
     shastraId *pSId;
{

  checkConn();
  sendReqString(REQ_QUIT, NULL);
  cmFlush(pHostKernel->fdSocket);

  kernelDisconnectHandler(pHostKernel->fdSocket);
}

int
```

```
kernelDisconnectHandler(fd)
     int fd;
{
  if(pHostKernel != NULL){
    fd = pHostKernel->fdSocket;

    listDestroy(pHostKernel->sendList, 1);
    listDestroy(pHostKernel->recvList, 1);
    free(pHostKernel);
    pHostKernel = NULL;

  }
  mplexUnRegisterChannel(fd);
}
```